LP-based approximation algorithms for Partial-Ordered Scheduling and Matroid Augmentation

Karl Christoph Heinz Walter Stickler

A thesis submitted for the degree of
Master of Philosophy

Department of Mathematics

London School of Economics
and Political Science

December 2023

Declaration

I certify that the thesis I have presented for examination for the MPhil degree of the London School of Economics and Political Science is my own work. The underlying research has been conducted in collaboration with my supervisor Dr Neil Olver and Dr Franziska Eberle for Chapter 2 and with my supervisor Dr Neil Olver, Prof. László Végh, and Dr Georg Loho for Chapter 3.

Abstract

In this thesis, we study two NP-hard problems from Combinatorial Optimization, from the perspective of approximation algorithms.

The first problem we study is called Partial-Order Scheduling on Parallel Machines, which we abbreviate to PO Scheduling. Here, we are given a partially ordered set of jobs which we want to schedule to a set of machines. Each job has some weight and some processing time associated to it. On each machine, the order of the jobs scheduled to it must agree with the given partial order, i.e., a job can only be started once all its predecessors scheduled to the same machine have been completed. However, two jobs scheduled to different machines are not constrained in any way. Thus, PO Scheduling deviates from the well-studied problem of precedence-constrained scheduling in this regard. The goal of PO Scheduling is to find a feasible schedule which minimizes the sum of weighted completion times of the jobs. PO Scheduling generalizes an already NP-hard version of scheduling introduced by Bosman, Frascaria, Olver, Sitters and Stougie [3], where they ask the same question as in PO Scheduling for the case where the jobs are totally ordered. The authors above present a constant-factor approximation algorithm for their problem. We conjecture that there is a constant-factor approximation algorithm for PO Scheduling as well. While we do not solve the problem, we give approximation algorithms for the special case that the partial order consists of disjoint totally ordered chains of linearly bounded length. Additionally, we give a structural result for optimal schedules in the case that the partial order consists of disjoint, backwardly ordered (with regard to the Smith-ratio) chains. We point towards some potential research directions.

For the Weighted Tree Augmentation Problem, we are given a graph with a distinguished spanning tree. Each non tree-edge has a cost associated to it. The goal is to find a cost-minimal set of edges such that when we add them to the tree-edges, the resulting graph is 2-edge-connected. Weighted tree augmentation is NP-hard. There has been recent progress in decreasing the best-known approximation factor for the problem by Traub and Zenklusen to $(1.5 + \varepsilon)$ [51, 52]. We study a generalization of weighted tree augmentation, called the Weighted Matroid Augmentation Problem, which we abbreviate to WMAP. In WMAP, we consider a matroid with a distinguished basis and a cost function on the non-basis elements. The goal is to find a cost-minimal set such that the union of the fundamental circuits of the elements in the set with regard to the distinguished basis cover that basis. We conjecture that there is a 2-approximation algorithm for the problem in the case that the matroid is regular. While we do not solve the problem, we give an approximation algorithm for the special case of the cographic matroid and show that there is no constant-factor approximation algorithm for WMAP for representable matroids unless $\mathsf{P} = \mathsf{NP}$.

Acknowledgements

I would like to thank the whole Department of Mathematics at LSE, and in particular my supervisor, Dr Neil Olver. Thanks a lot for the time spent discussing problems with me, for all information and research problems provided and for giving me an inside into the work and life of a member of academia. The same holds for my second supervisor Prof. László Végh, the entire Operations Research Group at the Department and for all members of the Department I have interacted and worked together with.

I want to especially thank the people I have worked together with on the problems which are presented in this thesis. For the problem of Partial-Order Scheduling on Parallel Machines, these are Dr Neil Olver and Dr Franziska Eberle. For the Weighted Matroid Augmentation Problem, these are Dr Neil Olver, Prof. László Végh and Dr Georg Loho. Thanks a lot for the time spent discussing together and trying to solve the problems presented. I also want to thank all other people who I've worked together with on different projects and everyone who supported me in writing this thesis by giving comments or by engaging in discussions about the material.

I would like to thank all PhD students and post-doctoral researchers for enhancing my stay in London and for forming an enriching social group at the Department.

Last, I also want to thank the NWO for providing me with financial support through a Vidi grant awarded to my supervisor Dr Neil Olver.

Contents

Chapter 1

Introduction

In this thesis, we consider designing constant-factor approximation algorithms for two NP-hard problems from the field of Combinatorial Optimization.

In Chapter 2 we discuss the problem of "Partial-Order Scheduling on Parallel Machines", which we abbreviate to PO Scheduling. We are given a set of machines M and a set of jobs J which is equipped with a partial order $\prec$. Each job $j \in J$ has an integral processing time p_j and a weight w_j associated to it. A schedule is an assignment of the jobs to time slots on machines such that a job j is assigned to p_j consecutive time slots on a single machine. The completion time C_j of j in a schedule is the last time slot j is assigned to. Each machine in each time slot can have at most one job assigned to it. The goal is to find a valid schedule where the sum of weighted completion times is minimized. For a schedule to be valid, besides the conditions stated above, on each machine the order of the completion times of the jobs scheduled to the machine must agree with the partial order $\prec$, i.e., when $j \prec k$ in the partial order and j and k are scheduled to the same machine, j must be completed before k can be started. Notably, if $j \prec k$ but j and k are scheduled to different machines, there is no constraint on their time slots relative to each other. This problem can be understood as a variation of precedence-constrained scheduling. In precedence-constrained scheduling, the restriction on start- and completion times of related jobs applies over all machines. Intuitively, we can interpret for example the setting of precedence-constrained scheduling as steps in an assembly line process. Before the final product can be assembled, each of the constituent parts must be completed, independent of the machine these parts are being processed. In the setting of PO Scheduling however, we can think of different jobs requiring different machine configurations. Thus, jobs have to be processed on each machine in a certain order, with some changes to the machine configuration between jobs. However, if two jobs are scheduled to two different machines, they do not constrain each other. Thus, while formally, precedence-constrained scheduling and PO Scheduling are quite close, there are significant differences in the set-up and its interpretation. Alternatively, PO Scheduling generalizes an already NP-hard problem considered by

Bosman, Frascaria, Olver, Sitters and Stougie [3] where they ask the same question as in PO Scheduling for the case where the jobs are totally ordered. We refer to this problem as TO Scheduling. The authors have presented a constant-factor approximation algorithm for TO Scheduling. Thus, we can understand our studies of PO Scheduling as an attempt to extend the result above for the case of a more general order.

We conjecture that there exists a constant-factor approximation algorithm for PO Scheduling. We give approximation algorithms for the case that the partial order consists of disjoint totally ordered chains of linearly bounded length. Additionally, we give a result on some structural properties of optimal schedules for the case where the partial order consists of disjoint totally ordered chains, with unit processing times which are ordered backwardly with regard to their weights, i.e., the weights of the jobs only increase along each chain.

In Chapter 3 we discuss the "Weighted Matroid Augmentation Problem", which we abbreviate as WMAP. The problem is inspired by recent progress made on the NP-hard weighted tree augmentation problem (WTAP) by Traub and Zenklusen [52]. In WTAP, we are given a graph $G = (V, E)$, a spanning tree $T \subseteq E$ and a cost function $c : E \backslash T \to \mathbb{R}$. The goal is to find a cost-minimal set $S \subseteq E \backslash T$ such that the graph $(V, T \cup S)$ is 2-edge-connected. This problem (and variations of it) are for example studied in the context of designing robust networks, when we want to ensure that a given network remains connected even if some links fail. Traub and Zenklusen improved the approximation factor on this problem below 2, making progress after a long time. We will see that WMAP is a generalization of WTAP. In WMAP, we are given a matroid $M = (E, \mathcal{I})$ with a distinguished basis B and a cost function $c : E \backslash B \to \mathbb{R}$. The goal is to find a cost-minimal set $S \subseteq E \backslash B$ such that the union of the fundamental circuits $C(s, B)$ of all $s \in S$ cover B.

We conjecture that there is a 2-approximation algorithm for WMAP for the case that the matroid is regular. We show that there is no constant-factor approximation algorithm for WMAP even for representable matroids unless $\mathsf{P} = \mathsf{NP}$. Next, we discuss how WTAP can be understood as a special case of WMAP, more specifically, how WTAP arises when we consider WMAP for the case of a graphic matroid. We re-contextualize two well-known 2-approximation algorithms for WTAP in terms of WMAP. We give a 2-approximation algorithm for WMAP for the case that the matroid is cographic. Based on the algorithms for these two cases, we discuss potential ways forward to tackle designing a 2-approximation algorithm for WMAP for regular matroids.

Definitions specific to the problems are discussed in the relevant chapters. For each of the problems, as mentioned, we give partial results, discuss what techniques we have deployed so far to try to solve the problems and what roadblocks we have encountered. We also state open directions which can be pursued to settle the conjectures. For now, we give a quick overview of approximation algorithms in

Section 1.1 and state relevant notation and definitions used throughout the thesis in Section 1.2.

1.1 Approximation Algorithms

As mentioned, in this thesis we consider finding approximation algorithms for two problems. We start by giving a quick introduction to the topic of approximation algorithms. A thorough overview of the field can be found in textbooks by Williamson and Shmoys [58] and Vazirani [57]. The definitions and notation used is this thesis follows mostly [58].

Many interesting problems which are studied in Combinatorial Optimization are NP-hard, including the ones we discuss in the following chapters. Thus, unless $\mathsf{P} = \mathsf{NP}$, we cannot expect to be able to solve all large instances of such problems to optimality, given our limited resources in processing power and speed. Therefore, a common approach is to search for a "reasonably good" solution, which we can find "fast". This is, we want to find a solution which we can compute in polynomial time in terms of the input size and which has cost which can be bounded in terms of the cost of the (unknown) optimum. Throughout the thesis, we assume, unless stated otherwise, that we are in a minimization setting, i.e., we are trying to find a solution that satisfies some constraints and minimizes some cost function[1]. This gives rise to the definition of an *approximation algorithm* [58][2]:

Definition 1. *Consider some optimization problem. Recall that we assume we are in a minimization setting. An α-approximation algorithm is an algorithm that runs in polynomial time in terms of the input size of each specific problem instance. Additionally, for all instances of the problem the algorithm produces a solution with cost that is bounded above by α times the value of an optimal solution.*

We call α the *approximation factor* of the approximation algorithm. For the problems discussed in this thesis, we give an overview in the respective chapters over how the best known approximation factor has developed over time by new approximation algorithms being introduced. When $\alpha \in O(1)$, i.e., when the approximation factor is a constant, we call the related algorithm a *constant-factor approximation algorithm*[3]. If there is a family of approximation algorithms such that for each $\varepsilon > 0$ there exists a $(1+\varepsilon)$-approximation algorithm, we call such a family of algorithms a *polynomial-time approximation scheme* (PTAS). Note that the run-time of such a

[1] Note that the definition can equivalently be formulated for a maximization setting [58].

[2] Note that some authors do not polynomial runtime as part of the definition of an α-approximation algorithm [24] and would refer to the object in Definition 1 e.g., as an α-approximation algorithm approximation algorithm with polynomial time bound. We however follow the [58] and [57] and include polynomially bounded runtime as part of the definition of an α-approximation algorithm.

[3] Recall again that we assume that we are in a minimization setting. For a maximization setting, we have to slightly reformulate our statements [58]

family of algorithms may depend on ε. However, for each individual algorithm ε is fixed and thus a constant.

However, some problems are hard to approximate. This can be quantified by assuming conjectures such as $\mathsf{P} \neq \mathsf{NP}$ and then e.g., showing that it is NP-hard to find an approximation algorithm with a certain approximation factor. Such results are called *inapproximability results*. An optimization problem which does not admit a PTAS (unless $\mathsf{P} = \mathsf{NP}$) is called APX-*hard*. Thus, for such a problem, there exists some lower bound $\beta > 1$ such that no approximation algorithm for the problem has an approximation factor better than β unless $\mathsf{P} = \mathsf{NP}$. When assuming stronger conjectures then $\mathsf{P} \neq \mathsf{NP}$, stronger inapproximability results can be achieved. One such assumption is the *Unique Games Conjecture* (UGC) [27]. The UGC states that it is NP-hard to find approximate values for certain types of games[4]. Using the assumptions above, it can be shown for certain optimization problems that they do not even admit a constant-factor approximation algorithm. We will encounter one such problem in Chapter 3.

Throughout the thesis, we assume that the reader has a basic understanding of standard notions from linear and integer optimization, i.e., is familiar with concepts such as a *linear program* (LP), an *integer program* (IP), algorithms for finding optimal solutions to such programs if they exist, the notion of *polytopes* and is familiar with the most relevant related results. A good overview of these topics can be found in a textbook by Schrijver [42]. A common way to design approximation algorithm proceeds by formulating the problem as an IP and then solving the linear relaxation, which means solving an LP. While solving an IP is hard (the decision-version of the problem is in NP), an LP can be solved in polynomial size in terms of its size (as long as the LP's size is polynomial in terms of the input size). The optimal value obtained from solving the LP (i.e., the linear relaxation) gives us a lower bound on the optimal value of the original problem. Approximation algorithms based on LPs often include a step where the optimal LP-solution is transformed or rounded to an integral solution to achieve a feasible solution to the original IP with cost which can be bounded by some suitable multiple of the cost of the LP-optimum. Alternatively, one can sometimes argue that the optimal LP-solution is already integral. We will encounter such LP-based approximation algorithms in Chapters 2 and 3.

1.2 Notation and Definitions

Let us now define some notation and state some definitions used throughout the thesis.

Let S be an arbitrary set. By $\mathcal{P}(S)$ we denote the *power set* of S, i.e., the set of all subsets of S. Let S, T be disjoint sets, i.e., $S \cap T = \emptyset$. We denote by $S \cup T$ the *disjoint union* of S and T. The disjoint union is the same as the union, i.e.,

[4]For the sake of brevity, we will not discuss the exact statement of the conjecture here. The details can be found in [27].

$S \dot\cup T = S \cup T$, we merely use this notation to emphasize that the two sets comprising the union are disjoint. By $|S|$, we denote the *cardinality* of S. Let $T \subseteq S$ be a subset of S. The *incidence vector* $\mathbb{1}(T) \in \{0,1\}^S$ is a vector with $\mathbb{1}(T)_i = 1 \Leftrightarrow i \in T$. We treat sets and incidence vectors interchangeably when convenient. Let $c, x \in \mathbb{R}^S$. We define $c(x) := \sum_{i \in S} c_i x_i$. Then, $c(T) := c(\mathbb{1}(T)) = \sum_{i \in T} c_i$. Let $c \in \mathbb{R}^S$ and $x \in \mathbb{R}^T$ with $T \subseteq S$. We define $c(x) := \sum_{i \in T} c_i x_i$.

A relation $\prec$ on a set S is called a *partial order-relation* (or simply a partial order), if for all $s \in S$ we have $\neg(s \prec s)$ and for all $s, t, u \in S$ we have if $s \prec t$ and $t \prec u$ then $s \prec u$ follows. Let $<$ be a partial order. We call $<$ a *total order* on S if in addition to the axioms above we have for all $s, t \in S$ that either $s < t$ or $s = t$ or $t < s$. For some partial order $\prec$ and for $s, t \in S$ we denote by $s \preceq t$ that either $s \prec t$ or $s = t$. Similarly, $s \leq t$ denotes that $s < t$ or $s = t$.

We denote by $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ the non-negative and positive real numbers. By $\mathbb{N}_{\geq 0}$ and $\mathbb{N}_{>0}$ we denote the non-negative and positive whole numbers (i.e., the integers in the latter case). Let $\mathbb{Z}$ be the set of the whole numbers. For $a \leq b$, $a, b \in \mathbb{Z}$, we denote the set of whole numbers between a and b by $[a, b] := \{n \in \mathbb{Z} \mid a \leq n \leq b\}$. We define $[a] := [1, a]$ for $a \in \mathbb{N}_{>0}$.

Throughout the thesis, we will assume that the reader has a basic understanding of standard notions from graph theory. A good overview can be found in a textbook by Bondy [2]. We include here some definitions related to graph theory used more frequently in this thesis for the sake of being self-contained. Let G be a graph. By $V(G)$ we denote the set nodes of G and by $E(G)$ the set of edges. If clear from context, we will drop the dependence on G, i.e., refer to V and E only. For some $k \in \mathbb{N}_{>0}$, G is *k-edge-connected* if for every pair of nodes $u, v \in V$ there are k edge-disjoint paths in E from u to v. Let $W \subseteq V$ be a subset of the nodes. The *node-induced subgraph* $G[W] := (W, E')$ with $E' := \{\{u, v\} = e \in E \mid u, v \in W\}$ is the subgraph of G with node set W and all edges with both endpoints in W. Let $F \subseteq E$ be a subset of the edges. The *edge-induced subgraph* $G[F] := (V', F)$ with $V' := \{v \in V \mid \exists e \in F : e = \{u, v\} \text{ for some } u \in V\}$ is the subgraph of G with edge set F and all nodes which are adjacent to edges in F. A subset $T \subseteq E$ of the edges is called a *spanning tree* if (V, T) is connected and does not contain any cycles. Let $r \in V$ be an arbitrary node. We can designate r as the root of a spanning tree T. Since in a tree, there is a unique path between every pair of nodes, for each node $u \in V$, we can define the *descendants* (nodes $v \in V$, where the unique path from r to v goes through u) and the *predecessors* (nodes $v \in V$, where the unique path from r to u goes through v) of u in terms of the tree T.

Chapter 2

Partial-Order Scheduling on Parallel Machines

In this chapter, we consider the problem of "Partial-Order Scheduling on Parallel Machines", which we abbreviate as *PO Scheduling*. In Section 2.1 we define PO Scheduling and describe its background, motivation, and relation to other problems in the wider area of scheduling problems. We conjecture that there is a constant-factor approximation algorithm for PO Scheduling. We show that PO Scheduling is a generalization of "Total-order Scheduling on parallel machines", or *TO Scheduling*, for which a constant-factor approximation algorithm in known. We give an introduction to TO Scheduling in Section 2.2 to introduce some definitions and techniques which will also prove relevant for PO Scheduling. In Section 2.3, we consider an important special case, the *Multiple-Chains Case*. We present results on approximation algorithms for the case when the length of each chain is linear in the amount of machines and give structural results on the optimal solutions. Last, in Section 2.4 we consider the general case of PO Scheduling. There, we present a different approach on how to tackle the problem of designing an approximation algorithm via the *Configuration IP*.

2.1 Problem Description and Motivation

Scheduling is a very well studied class of problems in Combinatorial Optimization. Usually, for scheduling we are given a set of *jobs* J with $n := |J|$ and a set of *machines* M with $m := |M|$. Each job $j \in J$ has a *processing time* p_j and often a *weight* w_j associated with it. We also have a set of *time slots* T. A *schedule* $S : J \to M \times T$ is an assignment of jobs to time slots on machines. Usually, there are some further constraints a schedule has to satisfy to be feasible. The goal is to find a feasible schedule that is optimal with regard to some given objective function. There is a vast variety of different scheduling problems both in terms of additional constraints imposed for a schedule to be feasible and in terms of the objective function. To

deal with the different cases, Graham, Lawler, Lenstra and Kan [20] introduced the *three-field notation* to denote different types of scheduling problems, which is still being updated to deal with new variations of scheduling being introduced[1].

A classic and well-studied type of scheduling problems is *precedence-constrained scheduling* [9, 38, 7, 22, 43]. Here, the job-set J is equipped with a partial order $\prec$. In a prominent version of the problem, each job $j \in J$ must be assigned to p_j consecutive time slots on a single machine (i.e., non-preemptive setting, no job-splitting). The job j can be scheduled to any machine and all machines are able to complete a job in the same amount of time p_j (i.e., parallel machines). We call the first time slot a job $j \in J$ is scheduled to the *start time* C'_j of j, and the last time slot j is scheduled to we call the *completion time* C_j of j. Note that $C_j = C'_j + p_j$ by our earlier requirements. If $j \prec k$ for some $j, k \in J$, the jobs must be scheduled in such a way that $C_j \leq C'_k$. The goal is to minimize the sum of weighted completion times $\sum_{j \in J} w_j {C_j}^2$. This problem is NP-hard [56] already for the case of *unit weights* and *unit processing times*, i.e., for $w_j = 1$ and $p_j = 1$ for all $j \in J$. Thus, a common approach is to design approximation algorithms to find near-optimal solutions. For the problem described above, the currently best known approximation factor is $2 + 2\ln(2) + \varepsilon$ for each $\varepsilon > 0$ [35]. A great survey on different versions of precedence-constrained scheduling and other related problem can be found in [6].

Next, we consider a variation of precedence-constrained scheduling we refer to as "Total-Order Scheduling on Parallel Machines", abbreviated as *TO Scheduling*. Here, the job set J is equipped with a total order $<$. Again, each job $j \in J$ must be scheduled to p_j consecutive time slots on a single machine, j can be scheduled to any machine and all machines are able to complete a job on the same amount of time p_j. If $j < k$ are scheduled to the same machine, we must again have that $C_j \leq C'_k$. However, if j and k are scheduled to different machines, there are no constraints, i.e., job k can be completed before job j is started. We say, on each machine, the order of the completion times of the jobs scheduled to the machine must *agree* with the order $<$. The of goal is again to minimize the sum of weighted completion times $\sum_{j \in J} w_j C_j$. We can understand TO Scheduling as a variation of precedence-constrained scheduling where the ordering of the jobs is only relevant on a per-machine basis. This problem was introduced by Bosman, Frascaria, Olver, Sitters and Stougie [3]. The authors present a constant-factor approximation algorithm for this NP-hard problem.

In this chapter, we consider a generalization of TO Scheduling. Instead of the

[1] We will not describe the notation in greater detail here, for details see [20]. PO Scheduling does not have a standardized three-field notation. We propose using *local-PO-prec* to describe the constraint-type, given the similarities to precedence-constrained scheduling. Thus, PO Scheduling as described would be denoted $P|\textit{local-PO-prec}|\sum_j w_j C_j$ in the extended three-field notation. For some of the scheduling problems mentioned, we give the three-field notation for the interested reader when applicable.

[2] The specific problem we describe here is referred to as $P|\textit{prec}|\sum_j w_j C_j$ in the three-field notation by [20].

job set J being equipped with a total order, we consider the case of the job set being equipped with a partial order $\prec$. Else, the same conditions as in TO Scheduling apply. In particular, on each machine, the order of the jobs scheduled to it given by the completion times must *agree* with the order $\prec$, as above. To give an example, assume $j \prec k$ and $j \prec l$ for some $j, k, l \in J$. Consider a feasible schedule which assigns jobs j and k to the same machine but job l to a different machine. Since $j \prec k$, j must be completed before k can be started, i.e., $C_j \leq C'_k$. However, it is possible for l to be completed before j is started since they are scheduled to different machines. We refer to this problem as "Partial-Order Scheduling on Parallel Machines" which we abbreviate as *PO Scheduling*. We denote an instance of PO Scheduling by $(J, \prec), M$. We conjecture that there is a constant-factor approximation algorithm for PO Scheduling.

Conjecture 2. *There is a constant-factor approximation algorithm for PO Scheduling.*

Thus, we want to extend the understanding gained by developing a constant-factor approximation algorithm for TO Scheduling [3] to the case of the set of jobs being equipped with a partial order instead of a total order.

2.1.1 Notes on the Setup and the Multiple-Chains Case

As mentioned above, there are different angles how TO Scheduling, and thus its generalization of PO Scheduling, can be motivated. Based on different points of view, different setups arise. To avoid confusion, we want to introduce two more or less natural settings and highlight their differences. We will do so by arguing based on a special case of PO Scheduling, which we call the *multiple-chains case*. While both setups can be extended to general partial orders[3], we think that the difference in motivation is most clear for the multiple-chains case.

For the multiple-chains case, consider an instance of PO Scheduling, when the partial order on J consists of multiple disjoint totally ordered chains. I.e., we can partition $J = J_1 \dot\cup J_2 \dot\cup ... \dot\cup J_l$ into l disjoint subsets. We have $J_i = \{j_{i_1}, j_{i_2}, ..., j_{i_{|J_i|}}\}$ with $j_{i_1} \prec j_{i_2} \prec ... \prec j_{i_{|J_i|}}$ for all $1 \leq i \leq l$. Also, $j \not\prec k$ if $j \in J_a, k \in J_b$ for $a \neq b$. As mentioned, we refer to this case as the *Multiple-Chains Case*. This setting is a natural case to consider since it is a generalization of TO Scheduling that seemingly keeps much of the original properties of the problem intact. The multiple-chains case will be a starting-point for some of the approaches described in this chapter. Thus, we spend some time discussing properties of this case and give some structural results for the multiple-chains case in Section 2.3.

Let us now consider the two different setups mentioned earlier for the example of the multiple-chains case.

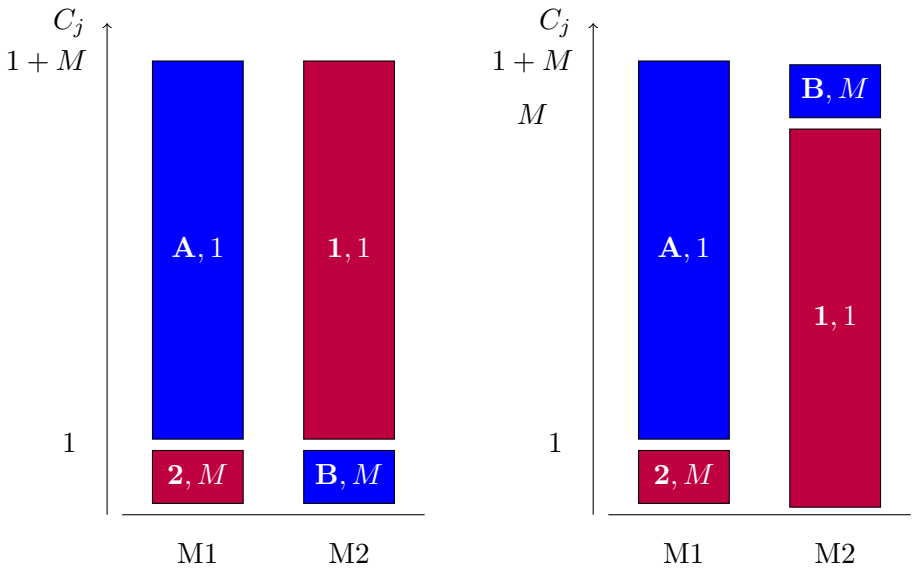
[3] In fact, the second setup we describe is just PO Scheduling as introduced earlier.

The first setup we call the *conveyor model*. Here, we interpret the chains as different conveyor belts on which jobs are presented. While all jobs arrive at the first time slot (i.e., no release-times), we can only assign a job to a machine once all predecessors from the same chain have already been assigned. Intuitively, in terms of a conveyor belt, we have to take care of the predecessors first, to access the later jobs. In this interpretation, where we think of assigning jobs to machines one at a time, the constraints of PO Scheduling arise naturally. However, is this setup, there are additional constrains on feasible schedules which are not present for PO Scheduling, as the problem was described above. As a consequence of the conveyor model, crucially, in any feasible schedule there must be a machine where the job scheduled to the first time slot is the first job of some chain (in terms of $\prec$). Thus, there are some additional constraints on what constitutes a feasible schedule for the conveyor model. Please consult Figure 2.1 for an example.

The second setup is PO Scheduling as we have introduced it before. Here, we don't interpret the different chains as conveyor belts but just as the abstract object of a partial order. Figure 2.1 shows an instance where the optimal solution for the conveyor-model and for PO Scheduling differ significantly.

Let us consider the instance with $J = \{1, 2, A, B\}$ where $1 \prec 2$ and $A \prec B$ form two disjoint chains. 1 and A are light jobs, $w_1 = w_A = 1$, with long processing times, $p_1 = p_A = M$, for some sufficiently large number $M \in \mathbb{N}_{>0}$. 2 and B are heavy jobs, $w_2 = w_B = M$, with short processing times, $p_2 = p_B = 1$. We have two machines, $m = 2$. In an optimal solution, we want to schedule 2 and B as early as possible. For PO Scheduling, we can schedule jobs 2 and A to the first machine in this order and jobs B and 1 to the second machine in this order. Effectively, we get rid of all constraints related to $\prec$ since at most one jobs of each chain is scheduled to each machine. We thus get an optimal value of $2(M \cdot 1) + 2\,(1 \cdot (1 + M)) = 4M + 2 \in O(M)$. In the conveyor model however, this solution wound not be feasible since as noted above not both jobs 2 and B can simultaneously be scheduled to the first time slots of the machines since they are "blocked" by jobs 1 and A. Thus, we could start by scheduling job 1 to the second machine. Then, we can access job 2 from that chain and schedule it to the first machine, to get it to the earliest possible time slot. Next, we can schedule job A on the first machine after job 2. Last, we can schedule job B to the second machine, after job 1. This gives us an optimal value of $M \cdot 1 + 1 \cdot (1 + M) + 1 \cdot M + M \cdot (1 + M) = M^2 + 4M + 1 \in O(M^2)$. In Figure 2.1, we depict the optimal solutions described above. The jobs $1, 2$ are depicted in purple, the jobs A, B are depicted in blue. Each job $j \in J$ is represented by a box with the **name of the job** (printed in bold type) and the weight w_j shown inside. The processing time p_j is indicated via the height of the box. On the "x-axis", we show the different machines and on the "y-axis" we show the completion times C_j for the schedule that is presented. Depictions similar to this will appear at different point in this chapter.

Thus, we see that we can reach qualitatively different results for the same instance in the different interpretations. Given these two potential models and their different



An optimal solution for PO Scheduling

An optimal solution in the conveyor model

Figure 2.1

behaviors, we want to clarify that throughout this chapter we focus on PO Scheduling as initially defined. However, the conveyor model might also be an interesting setup to be studied.

Given the setup described so far, before diving into the possible approaches on how to design an approximation algorithm for PO Scheduling, in Section 2.2 we give a quick overview of the methods used to design a constant-factor approximation algorithm for TO Scheduling. Some of the concepts presented there will be used there later when working with PO Scheduling. In Section 2.2, we also establish some notation used throughout this chapter.

2.2 Total-Order Scheduling on Parallel Machines

In this section, we give a brief overview of the constant-factor approximation algorithm for the problem of TO Scheduling as presented by Bosman, Frascaria, Olver, Sitters and Stougie [3]. We state the major definitions, as many of them will later be adapted to the partial order setting. We also introduce some of the most important results, however for the sake of brevity, we do not give any proofs. These can of course be found in the paper [3].

The main theorem of the paper is that there is a constant-factor approximation algorithm for TO Scheduling.

Theorem 3 ([3])**.** *There is a constant-factor approximation algorithm for TO scheduling with an approximation factor of* $\frac{27}{2} + 9\sqrt{3} < 29.1$.

To prove Theorem 3, the authors first show that we only lose a constant factor when looking for certain types of schedules, which we will describe in the following, compared to the optimum.

Recall the definition of TO Scheduling from Section 2.1 and let S be a feasible schedule for TO Scheduling on some input with jobs J and machines M, where J is equipped with a total order $<$. Let $n := |J|$ and $m := |M|$. Since J is totally ordered, we can identify $J := [n]$ and assume that $<$ is the natural order on $[n]$. Similarly, we identify $M := [m]$. For a job $j \in J$, we denote by $S(j) \in M$ the machine j is scheduled to. For a job j with weight w_j and processing time p_j we define the *Smith-ratio*[4] $\rho_j := \frac{w_j}{p_j}$. We assume that all Smith-ratios ρ_j are positive powers of some $\gamma \in (0, 1)$. This can always be achieved by rounding and scaling the weights and only loses us a factor of at most $\frac{1}{\gamma}$ in the approximation compared to the optimum. In case not all Smith-ratios are positive powers of γ, let w'_j and ρ'_j be the values before rounding. Set w_j such that ρ_j is the power of γ which is closest to ρ'_j from above. We thus have $w_j \leq \frac{1}{\gamma} w'_j$. We can now scale all weights such that the Smith-ratios are all positive powers of γ. This only loses us the factor mentioned above by rounding and scaling the weights as described. Note that this procedure is not limited to TO Scheduling but works e.g., for PO Scheduling as well.

Intuitively, a job with high Smith-ratio should be scheduled to an early time slot, as the job has high weight compared to its processing time. We define a new order $<_{SM}$ via $j <_{SM} k$ if and only if $j < k$ and $\rho_j \leq \rho_k$. Therefore $<_{SM}$ is a partial order where jobs are only comparable when they are comparable in the original order and when they are *backwards* with regard to their Smith-ratios. Intuitively, we ignore the original order when jobs are already in the right order with regard to the Smith-ratio, i.e., when the job with the higher Smith-ratio comes first. We call the new order $<_{SM}$ the *Smith-order*.

A schedule S is *Smith-monotone* if, for every $j <_{SM} k$, it holds that $S(j) \leq S(k)$. First, we consider the case of TO Scheduling with unit processing times, i.e., $p_j = 1$ for all $j \in J$. We say a schedule S is *staircase shaped* if for every prefix of the jobs in terms of the total order $<$, i.e., for $1 < 2 < ... < k$ for some $k \leq n$, the number of jobs assigned to each machine decreases monotonically with the machine index. In other words, every prefix of jobs is assigned to machines in such a way that machines with higher index receive at most as many jobs as machines with lower index. Thus, in a stair-case shaped schedule, the jobs $1 < 2 < ... < k$ form a "decreasing staircase" where machine β has at most as many jobs from $[1, k]$ assigned to it as has machine α with $\alpha \leq \beta$. Equivalently, we can say that for each job $j \in J$ that $S(j) = |\{k \in J \mid k < j, C_j = C_k\}| + 1$ must hold.

[4]Based on the work of Smith [50], who conducted much of the early research on scheduling problems.

Lemma 4 ([3]). *For unit processing times, there exists an optimal schedule that is Smith-monotone and staircase shaped.*

Note that the concept of a staircase-shaped schedule is only defined for the case of unit-processing times. For general processing times, an equivalent statement to Lemma 4 does not hold. However, by focusing on Smith-monotone schedules, we only lose a constant factor compared to the optimum. Let S be an optimal Smith-monotone schedule and let $S^\star$ be an optimal schedule for our instance of TO-Scheduling. By $w(S)$ we denote the cost of the schedule, i.e., the sum of weighted completion times.

Lemma 5 ([3]). *An optimal Smith-monotone schedule S has cost $w(S)$ at most $\frac{3}{2}w(S^\star)$.*

Thus, we can focus on finding an optimal Smith-monotone schedule. In the rest of this section, we focus on TO Scheduling in general, i.e., we won't assume unit processing times.

Next, the authors introduce the *partial completion time* $\tilde{C}_j$ of a job $j \in J$ as $\tilde{C}_j := \sum_{k \leq_{SM} j : S(k)=S(j)} p_k$. Thus, the partial completion time $\tilde{C}_j$ of a job $j \in J$ is the completion time we would achieve if we only considered the jobs on the same machine which are earlier with regard to the total order and which have a smaller Smith ratio. The authors then consider changing the objective function by using the partial completion time instead of the standard completion time. They show that both objective values are within a constant factor of each other.

Theorem 6 ([3]). *Take an instance where all Smith ratios are positive powers of $\gamma \in (0,1)$. Consider any schedule S. It holds that*

$$\sum_{j \in J} w_j \tilde{C}_j \leq \sum_{j \in J} w_j C_j \leq \left(\frac{4}{1-\sqrt{\gamma}} - 3 \right) \sum_{j \in J} w_j \tilde{C}_j$$

Given Theorem 6 and from what we have seen so far, we can focus on finding a Smith-monotone schedule which is optimal under the partial completion time while assuming that all Smith ratios are positive powers of some $\gamma \in (0,1)$.

The problem above can be formulated as an IP. For each job $j \in J$, we have an integer variable $u_j \in [m]$ where $u_j = \alpha$ indicates that j should be scheduled to machine α. Additionally, for each pair of jobs $j, k \in J$ we have a binary variable z_{kj} where $z_{kj} = 1$ indicates that j and k are scheduled to different machines. We obtain the following IP:

$$\begin{array}{llcll} \min & \sum_{j \in J} w_j \tilde{C}_j & & & \\ \text{s.t.} & \tilde{C}_j & \geq & p_j + \sum_{k <_{SM} j} (1 - z_{kj}) p_k & \forall j \in J \\ & u_j & \geq & u_k + z_{kj} & \forall k <_{SM} j, j, k \in J \\ & u_j & \in & [m] & \forall j \in J \\ & z_{kj} & \in & \{0,1\} & \forall k <_{SM} j, j, k \in J \end{array} \tag{2.1}$$

Note that the IP 2.1 does not explicitly encode which time slot each job is scheduled to. However, we can still obtain a valid schedule from the solution since we know which machine each job is scheduled to while on each machine the order of jobs is dictated by the total order $<$. Consider the linear relaxation of the IP 2.1, where we replace the integrality constraints by $1 \leq u_j \leq m$ for all $j \in J$ and by $0 \leq z_{kj} \leq 1$ for all $k <_{SM} j$, where $j, k \in J$. Since we have polynomially many constraints, we can find an optimal solution to the linear relaxation in polynomial time. Let $(z^\star, u^\star, \tilde{C}^\star)$ be such an optimal solution. It remains to round this solution to a feasible integral solution.

To do so, the authors introduce *β-point rounding*. β-point rounding can be understood as a variation of α-point rounding, which has been applied in different setting to scheduling problems [45]. In β-point rounding, for $\beta \in (0, 1)$, the *β-point* associated to $u^\star$ is the schedule obtained by scheduling job $j \in J$ to machine $\lceil u_j^\star - \beta \rceil$ and sorting the jobs on each machine according to the total order. β is chosen uniformly at random from $(0, 1)$, making the β-point schedule a random schedule. In Proposition 7 the authors show that rounding the schedule in expectation does not increase cost. This means that there must exist an integral solution with cost at most the cost of the optimum $(z^\star, u^\star, \tilde{C}^\star)$ of the linear relaxation, which can be found by applying β-point rounding. The authors also discuss how the process can be derandomized.

Proposition 7 ([3])**.** *For any $j \in J$ we have that $\mathbb{E}[\tilde{C}_j] \leq \tilde{C}_j^\star$. Thus*

$$\mathbb{E}\left[\sum_{j \in J} w_j \tilde{C}_j\right] \leq \sum_{j \in J} w_j \tilde{C}_j^\star.$$

Combining the statements from Lemma 5, Theorem 6 and Proposition 7, the authors can prove the main Theorem 3 showing that there is a constant-factor approximation algorithm for TO Scheduling.

After having seen the main concepts employed to design a constant-factor approximation algorithm for TO Scheduling, in the next Section 2.3 we return to the problem of PO Scheduling. We start by considering the multiple-chains case, for which we will make use of many of the concepts described in this section.

2.3 Results for the Multiple-Chains Case

In this section, as a first step for designing an approximation algorithm for PO Scheduling, we try to find an approximation algorithm for the multiple-chains case. This is the special case of PO Scheduling where the partial order consists of multiple disjoint totally ordered chains. I.e., we can partition $J = J_1 \cup J_2 \cup ... \cup J_l$ into l disjoint subsets. We have $J_i = \{j_{i_1}, j_{i_2}, ..., j_{i_{|J_i|}}\}$ with $j_{i_1} \prec j_{i_2} \prec ... \prec j_{i_{|J_i|}}$ for all $1 \leq i \leq k$. Also, $j \not\prec k$ if $j \in J_a, k \in J_b$ for $a \neq b$. Additionally, we assume throughout this section that we have unit processing times, i.e., $p_j = 1$ for all $j \in J$.

We approach designing an approximation algorithm by deriving a result on the structure of optimal schedules in Section 2.3.2 which then can be used to give an IP-formulation for the multiple-chains case in Section 2.3.3. We will see that the structural result does only add sufficient information for chains from a certain length onwards. Thus, we discuss "short chains" separately in Section 2.3.1. We show that we can find separate approximation algorithms for such short chains, independent of the IP introduced in Section 2.3.3.

2.3.1 Approximation Algorithms for Short Chains

Let $(J, \prec), M$ be an instance of PO Scheduling where $\prec$ consists of l disjoint totally ordered chains. Let $|J| = n$ and let $|M| = m$. For $1 \leq i \leq l$ let J_i denote the i^{th} chain. Recall that we assume $p_j = 1$ for all $j \in J$. We consider the case where all chains have length linear in the number of machines, i.e., when $|J_i| \leq \alpha m$ for all $1 \leq i \leq l$ and for some natural number α. We call this case the *Short Chains Case*.

We first consider the short chains case where $\alpha = 1$, i.e., when $|J_i| \leq m$ for all $1 \leq i \leq l$. Given this, we can consider the following Algorithm.

Algorithm 1: 2-Approximation Algorithm for Short Chains Case for $\alpha = 1$

Input : $(J, \prec), M$ with $p_j = 1$ for all $j \in J$ and where $\prec$ consists of disjoint totally ordered chains each of length at most m.

Output : A feasible schedule of J with cost at most twice the optimum.

```
1 Relabel the jobs by [n] such that w_j ≥ w_k for all j ≤ k.
2 for j = 1 to n do
3 |   Consider j and let J_i be the chain that j belongs to.
4 |   Schedule j to the earliest free time slot on the machine with lowest
  |     current load which doesn't already have a job of chain J_i scheduled
  |     to it.
5 end
6 Let S be the schedule thus obtained. Return S.
```

Algorithm 1 assigns jobs one-by-one non-increasingly by weight to the machine with the smallest number of jobs currently scheduled to it which doesn't have a job from the same chain already scheduled to it. Since there are at most m jobs per chain and m machines, we can always find a suitable machine. Thus, on each machine, no two jobs are comparable. We will see that we can bound the completion time of a job being scheduled compared to the case where we spread the jobs evenly over all machines, ignoring all constraints related to the order $\prec$.

Lemma 8. *Algorithm 1 is a 2-approximation algorithm for the short-chains case for $\alpha = 1$.*

Proof. First, we show that the algorithm runs in polynomial time in terms of the

input size. We can relabel the jobs decreasingly by weight and breaking ties arbitrarily in polynomial time. All other steps can be completed in polynomial time as well. Since the main loop is only executed n times, the algorithm overall runs in polynomial time.

Next, we show that the solution returned is feasible. Consider some job $j \in J_i$, for $1 \leq i \leq l$, which is currently to be scheduled by the algorithm. Since there are m machines but only at most m jobs per chain, we can always find some machine with no job of chain J_i already scheduled to it. Thus, on each machine, there is at most one job of each chain and thus no two jobs on the same machine are comparable under $\prec$. Therefore, we trivially have that the constraints related to the order $\prec$ are satisfied, i.e., on each machine the order of completion times of the comparable jobs agrees with the order $\prec$.

Last, we show that the solution returned has the claimed cost. Let S be the schedule obtained from the Algorithm 1 on the input $(J, \prec), M$ and let C_j be the completion time of job $j \in J$ under S. To bound the cost of the schedule S, we first consider an optimal schedule $S^{\emptyset}$ of the instance $(J, \emptyset), M$, i.e., our original instance where we don't have any order constraints. Let $S^{\star}$ be the optimal schedule for our current instance $(J, \prec), M$ and let $C_j^{\emptyset}$ and $C_j^{\star}$ be the completion times of $j \in J$ under $S^{\emptyset}$ and $S^{\star}$ respectively. Since $S^{\star}$ is feasible for the instance $(J, \emptyset), M$, we have $\sum_{j \in J} w_j C_j^{\emptyset} \leq \sum_{j \in J} w_j C_j^{\star} \leq \sum_{j \in J} w_j C_j$. We claim that $C_j \leq 2C_j^{\emptyset}$, from which the lemma would then follow.

Consider some job $j \in J$ for $1 \leq j \leq n$. Note that j is the j^{th} job to be scheduled since we have relabeled the jobs accordingly. We have $C_j^{\emptyset} = \left\lceil \frac{j}{m} \right\rceil$ since for $(J, \emptyset), M$ we want to put the jobs with the highest weights as early as possible, which we can do since we don't have any order constraints[5]. Consider the partial schedule S' of all jobs up to j obtained by the algorithm, i.e., the partial schedule obtained by the algorithm just after we have scheduled job j. Let L_i be the last occupied time slot of machine $1 \leq i \leq m$ in S'. We re-label the machines such that $L_1 \leq L_2 \leq ... \leq L_m$. Let i be the machine job j has been assigned to by the algorithm. Thus, $L_i = C_j$. Note that we must have $L_1 \leq C_j^{\emptyset}$ since machine 1 currently has minimal load and since only j jobs are currently scheduled in S'. Now, consider some machine α and some job $k \in J^{\alpha}$, $j \neq k$, which is already scheduled in S' with $C_k > C_j^{\emptyset}$[6]. Let $k \in J_a$ be in chain J_a for some $a \in [l]$. Then, there must be a job of the same chain J_a scheduled to machine 1 as else job k would have been scheduled to machine 1 by the algorithm. Since this must hold for each job with the same characteristics on machine α, we must have $L_{\alpha} \leq C_j^{\emptyset} + L_1$. Thus, putting the arguments above together, we can bound the current completion time of j in S' and thus in S as

[5]Note that $(J, \emptyset), M$ describes scheduling without precedence constraints on parallel machines. $C^{\emptyset}$ is the same schedule as obtained from applying the well-known greedy algorithm "list scheduling" (see e.g., [19]). In list scheduling, we assign jobs by non-increasing weight to the earliest free time slot. In the setting we have described it is well known that list scheduling yields an optimal schedule.

[6]If no such job exists, we have $C_o \leq C_o^{\emptyset}$ of all $o \in J$ and the statement of the lemma follows trivially.

follows: $C_j = L_i \leq C_j^{\emptyset} + L_1 \leq 2C_j^{\emptyset}$. Therefore, the overall statement follows. □

Next, we consider the general short chains case, i.e., where $|J_i| \leq \alpha m$ for all $1 \leq i \leq l$ for some natural number α. Let $(J, \prec)$ be the given job set, where $J = J_1 \dot{\cup} J_2 \dot{\cup} ... \dot{\cup} J_l$. We consider a new job set $(J', \prec')$ where we combine tuples of α successive jobs in a chain into a new single job. More formally, consider jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b} \in J_i$ for some $1 \leq i \leq l$ with $a = \alpha c + 1$, $b = \min\{\alpha(c+1), |J_i|\}$ for some $1 \leq c \leq m$. We combine this set of at most α jobs into a single job j'_{i_c} with $p_{j'_{i_c}} = 1$ and $w_{j'_{i_c}} = \sum_{d=a}^{b} w_{j_{i_d}}$. Thus, we obtain a set $J' = J'_1 \dot{\cup} J'_2 \dot{\cup} ... \dot{\cup} J'_l$ with $J'_i = \{j'_{i_1}, j'_{i_2}, ..., j'_{|J'_i|}\}$ and $|J'_i| = \left\lceil \frac{|J_i|}{\alpha} \right\rceil \leq \left\lceil \frac{\alpha m}{\alpha} \right\rceil = m$ for $1 \leq i \leq l$. We define a new order $\prec'$ on J' as follows: For $J'_i = \{j'_{i_1}, j'_{i_2}, ..., j'_{|J'_i|}\}$ we have $j'_{i_1} \prec' j'_{i_2} \prec' ... \prec' j'_{|J'_i|}$. For $j' \in J'_a$, $k' \in J'_b$ with $a \neq b$ we have $\neg (j \prec' k)$. Intuitively $\prec'$ is a contracted version of $\prec$ after combining the tuples of jobs as described above. Note that $(J', \prec'), M$ is an instance of the short chains case for $\alpha' = 1$. Given this, for the (general) short chains case, we can state Algorithm 2:

Algorithm 2: 2α-Approximation Algorithm for the (general) short chains case

Input : $(J, \prec), M$ with $p_j = 1$ for all $j \in J$ and where $\prec$ consists of disjoint chains each of length at most αm for some fixed constant α.

Output : A feasible schedule of J with cost at most 2α the optimum.

1 Compute $(J', \prec')$ as described in Subsection 2.3.1.
2 Let S' be the output of Algorithm 1 of the input $(J', \prec'), M$.
3 Consider $j' \in J'$. Let $j_a, j_{a+1}, ..., j_b \in J$ be the jobs which were combined to form j'. Denote by $C_{j'}$ be the completion time of j' in S'. Let S be the schedule obtained from S' by scheduling the jobs $j_a, j_{a+1}, ..., j_b$ according to the order $\prec$ in the time slots from $(\alpha - 1)C_{j'} + 1$ to $\alpha C_{j'}$ on the same machine j' was scheduled to in S'.
4 Return S.

Intuitively, on each chain, the algorithm combines each set of α consecutive jobs into a single job with processing time 1. As the original chain has at most αm jobs, the new chain has at most m jobs. Thus, we can apply Algorithm 1 on the new instance with the combined jobs. Intuitively, we show that combining each set of at most α consecutive jobs only loses us a constant factor of α compared to the objective value of the optimal solution. The rest follows from the proof of Lemma 8.

Theorem 9. *Algorithm 2 is a 2α-approximation algorithm for the (general) short chains case.*

Proof. We can compute $(J', \prec')$ in polynomial time from $(J, \prec)$. As noted, $|J'_i| = \left\lceil \frac{|J_i|}{\alpha} \right\rceil \leq \lceil \frac{\alpha m}{\alpha} \rceil = m$ for all $1 \leq i \leq l$. Thus, $(J', \prec'), M$ is an instance of the short chains case with $\alpha' = 1$. Algorithm 1 runs in polynomial time as shown in Lemma 8. Let S' be the output of Algorithm 1 on the input $(J', \prec'), M$. We can obtain S from S' in polynomial time. Thus, the algorithm runs in polynomial time.

By construction of J', at most α different jobs of J are combined into a single job j'. Thus, we give ourselves enough time slots when constructing S from S' to schedule all jobs in J which were combined to form j'. In S', we have at most one job from each chain per machine. Thus, in S we have at most α jobs from each chain per machine. By construction, all these jobs are scheduled according to the order $\prec$. Therefore, S is a feasible schedule.

Let C_j denote the completion time of a job j in S. Let $S^\emptyset$ be an optimal schedule for the instance $(J, \emptyset), M$, i.e., our original instance but without any order constraints and let $C_j^\emptyset$ be the completion time of j in $S^\emptyset$. Let $S^\star$ be an optimal schedule for the instance $(J, \prec), M$, where $C_j^\star$ denotes the completion time of $j \in J$. As $S^\star$ is a feasible schedule for the instance $(J, \emptyset), M$, we have that $\sum_{j \in J} w_j C_j^\emptyset \leq \sum_{j \in J} w_j C_j^\star \leq \sum_{j \in J} w_j C_j$. We claim that $C_j \leq 2\alpha C_j^\emptyset$ for all $j \in J$ from which the lemma would then follow.

Fix a job $j \in J$. Consider the jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b} \in J_i$ for some $1 \leq i \leq l$ with $a = \alpha c + 1$, $b = \min\{\alpha(c+1), |J_i|\}$ for some $1 \leq c \leq m$ such that we have $j = j_{i_\gamma}$ for some $\gamma \in [a, b]$. Let $j' \in J'$ be the job that was obtained by combining jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b}$. Let $\bar{S}$ be an optimal schedule for $(J', \emptyset), M$, i.e., for the instance with jobs J' without any ordering constraints. Let $\bar{C}_{j'}$ be the completion time of j' under $\bar{S}$ and let $C'_{j'}$ be the completion time of j' under S'. S' is the schedule computed by Algorithm 2 in line 2. Note that by the same arguments as in the proof of of Lemma 8, we obtain $C'_{j'} \leq 2\bar{C}_{j'}$, as S' is a schedule obtained by Algorithm 1 on the input $(J', \prec'), M$, while $\bar{S}$ is an optimal schedule for $(J', \emptyset), M$. Now, consider the schedule $\tilde{S}$ for the jobs J obtained from $\bar{S}$ by scheduling the jobs $j_a, j_{a+1}, ..., j_b$ according to the order $\prec$ in the time slots from $(\alpha - 1)\bar{C}_{j'} + 1$ to $\alpha \bar{C}_{j'}$ on the same machine as j' was scheduled to in $\bar{S}$. We denote by $\tilde{C}_j$ the completion time of j under $\tilde{S}$. Note that $\tilde{S}$ and S are obtained from $\bar{S}$ and S' in the same manner. Thus, we obtain that $C_j \leq 2\tilde{C}_j$.

Consider again the schedule $S^\emptyset$ which is optimal for $(J, \emptyset), M$. Consider any tuple of jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b} \in J_i$ for some a, b as described above. Note that here we do not require that $j = j_{i_\gamma}$ for some $\gamma \in [a, b]$, i.e., we just consider an arbitrary tuple of jobs which are combined into a single job in J'. For these jobs, we define $\omega_{a,b} = \min\left\{\alpha C^\emptyset_{j_{i_\delta}} \mid \delta \in [a, b]\right\}$. Let $\delta_{a,b}$ be a job from the tuple of jobs above such that $\alpha C^\emptyset_{\delta_{a,b}} = \omega_{a,b}$. Let $\bar{S}^\emptyset$ be the schedule where we schedule each such tuple of jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b}$ in order according to $\prec$ to the time slots from $\omega_{a,b} - \alpha + 1$ to $\omega_{a,b}$ to the machine job $\delta_{a,b}$ is scheduled to in $S^\emptyset$. We can thus understand $\bar{S}^\emptyset$ as being obtained by first multiplying each completion time in $S^\emptyset$ by α and then scheduling each tuple of jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b}$ to the α time slots before and including $\omega_{a,b}$, i.e., the earliest

time slot any of these jobs is completed in the scaled-up version of $S^{\emptyset}$. Let $\bar{C}_j^{\emptyset}$ be the completion time of job j in $\bar{S}^{\emptyset}$. By construction, we have $\bar{C}_j^{\emptyset} \leq \alpha C_j^{\emptyset}$. Additionally, by definition of $\bar{S}$ (as an optimal schedule to $(J', \emptyset), M$) and subsequently of $\tilde{S}$ (as described above), we have $\tilde{C}_j \leq \bar{C}_j^{\emptyset}$. This is since $\bar{S}^{\emptyset}$ can be understood as a feasible schedule for the instance $(J, \emptyset), M$ with the additional constraint that all such tuples of jobs $j_{i_a}, j_{i_{a+1}}, ..., j_{i_b} \in J_i$ as above (i.e., tuples of jobs which would be combined into a single job in J') must be scheduled to subsequent time slots, while $\tilde{S}$ is an optimal schedule for this problem. Putting everything together, we obtain $C_j \leq 2\tilde{C}_j \leq 2\bar{C}_j^{\emptyset} \leq 2\alpha C_j^{\emptyset}$. Thus, the lemma follows. □

Overall, we have therefore seen that we can find constant-factor approximation algorithms for the short chains case for each fixed α.

2.3.2 A Structural Result on Optimal Schedules for Multiple-Chains

Next, we consider chains of arbitrary lengths. We still assume unit processing times, $p_j = 1$ for all $j \in J$. Additionally, we impose that chains are ordered *backwardly*, meaning that among each chain, the weights of the jobs only strictly increase along with increasing elements in the chain according to $\prec$. Thus, we can partition $J = J_1 \cup J_2 \cup ... \cup J_l$ into l disjoint subsets. We have $J_i = \{j_{i_1}, j_{i_2}, ..., j_{i_{|J_i|}}\}$ with $w_{i_1} < w_{i_2} < ... < w_{i_{|J_i|}}$ for all $1 \leq i \leq k$. To justify why we start by focusing on backwardly ordered chains, consider the following: Intuitively, to minimize the sum of weighted completion times, we would like to schedule jobs with high weight as early as possible. Under our assumption, jobs with higher weight are "blocked" as much as possible by jobs with lower weight in front of them. Thus, studying this case might generate insights into how to develop approximation algorithms for more general cases. We think it is likely that the results in this section can be extended to the case of non-decreasing weights, i.e., when adjacent weights can be equal, by adjusting the arguments below by introducing a secondary, auxiliary potential function, similar to as was done in [3] in the proof of Lemma 1 (in their paper). We leave incorporating such an improved potential function open for now.

We show in Lemmas 11, 12 and 13 as well as in Corollary 14 that under the assumptions described above, on each machine in an optimal schedule all jobs of the same chain are scheduled to consecutive time slots. Furthermore, each chain changes machines at most $2m - 2$ times (Corollary 14). Note that we consider a quite specific special case of PO Scheduling. It might be interesting to determine the complexity of this problem, i.e., showing that the problem is either in P or is NP-complete. We haven't focused on this question too much and leave it open for now.

The results stated in this section rely on Algorithm 3 by Sidney [48]. Sidney's algorithm is an algorithm for precedence-constrained scheduling on a single machine. We will shortly point out how the algorithm is related to PO Scheduling. We give a

slightly adjusted version of the original algorithm and the related result, which are more suitable to our setting. Specifically, the original algorithm by Sidney does not require unit processing time, which we assume for convenience.

Let $(J, \prec), \{1\}$ be an instance of precedence-constrained scheduling of jobs J to a single machine. Assume that $p_j = 1$ for all $j \in J$. Let $V \subseteq J$. We define $w(V) := \sum_{j \in V} w_j$ and $\rho(V) := \frac{w(V)}{|V|}$. I.e., $\rho(V)$ denotes the Smith-ratio or equivalently the average weight of V. We say $U \subseteq V$ is *initial* for V if for all $j \in U$ there is no $k \in V \backslash U$ such that $k \prec j$. $U \subseteq V$ is *w-maximal* for V if U is initial for V and $\rho(U) \geq \rho(W)$ for any $W \subseteq V$ which is initial for V. Lastly, $U \subseteq V$ is *$w^\star$-maximal* for V if U is w-maximal for V and if there is no $W \subsetneq U$ such that W is w-maximal for V. In terms of a single totally-ordered chain, U is $w^\star$-maximal for V if it is a prefix of jobs in V which has minimal size among the prefixes of maximal average weight. Thus, we can introduce Algorithm 3 by Sidney [48].

Algorithm 3: Sidney's Algorithm [48]

Input : $(J, \prec)$ with $p_j = 1$ for all $j \in J$, an instance of precedence-constrained scheduling on a single machine.

Output : An optimal schedule.

```
1 Set S ← ∅.
2 Set V ← J.
3 while V ≠ ∅ do
4 |   Find U that is w*-maximal for V.
5 |   Let S(U) be an optimal schedule of U. Append S(U) to S.
6 |   Set V ← V\U.
7 end
8 Return S.
```

Sidney [48] has shown a result about the optimality of schedules for precedence-constrained scheduling on a single machine, Theorem 10. Note that the original result does not assume unit processing times. We consider the case of unit processing times here for notational convenience.

Theorem 10 ([48])**.** *A schedule S is optimal for precedence-constrained scheduling with jobs $(J, \prec)$ on a single machine and unit processing times if and only if S can be generated by Algorithm 3.*

Note that for the case of a general partial order, it is not clear how to find a $w^\star$-maximal subset of jobs or an optimal schedule for such a subset in polynomial time. However, for the case that the order consists of disjoint chains both problems become much more straight forward.

Given Algorithm 3, we can now start discussing the structure of optimal schedules for the multiple-chains case when all chains are ordered backwardly. Consider an instance $(J, \prec), M$ of PO Scheduling satisfying all assumptions stated before. Let

S be some feasible schedule. For some chain J_i, $1 \leq i \leq l$, and some machine $a \in M$ we denote by $T_S(i,a) := \{t \mid C_j = t, S(j) = a \text{ for some } j \in J_i\}$ the set of time slots jobs of chain J_i on machine a are scheduled to. We drop the dependence on S when the reference is clear. Let $S^\star$ be an optimal schedule to our instance $(J, \prec), M$.

Lemma 11. *For all J_i, $1 \leq i \leq l$ and $a \in M$ we have $T_{S^\star}(i,a) = [\alpha, \beta]$ for some suitable α, β. I.e., all jobs from the same chain which are scheduled to the same machine occupy successive time slots.*

Proof. Consider the set of jobs $J^a := \{j \in J \mid S^\star(j) = a\}$ which is scheduled to machine a by $S^\star$. For $S^\star$ to be optimal, J^a must be scheduled optimally, as else we could replace the schedule on machine a with an optimal schedule of J^a and thereby decrease the sum of weighted completion times, contradicting the optimality of $S^\star$. Thus, when considering how J^a is scheduled (as part of an optimal schedule $S^\star$), we can consider PO Scheduling of only the jobs in J^a to a single machine. Note that PO Scheduling on a single machine is identical to precedence-constrained scheduling with the same order $\prec$ on a single machine, as the difference between both problems is only manifest when multiple machines are involved. Therefore, Theorem 10 must hold for the schedule of J^a (as part of the schedule $S^\star$), i.e., the schedule of J^a must have been generated by Algorithm 3.

Let $J_i^a := J^a \cap J_i$. To conclude the proof, it suffices to show that only some set J_i^a would be chosen by Algorithm 3 to be scheduled next. I.e., no set of jobs W which intersects at least two chains would ever be chosen and no proper prefix of jobs of a single chain $V \subsetneq J_i^a$ would ever be picked by Algorithm 3 to be scheduled next.

For the first case, let $W = \bigcup_{\gamma \in \Gamma \subset [l]} J_\gamma^a$ where for each $\gamma \in \Gamma$ we have $W_\gamma := W \cap J_\gamma^a \neq \emptyset$ and $|\Gamma| \geq 2$, i.e., W intersects at least two chains. Let i, j be the chain-indices maximizing $\rho(W_\gamma)$, where we break ties arbitrarily. W can only be w-maximal if $\rho(W_i) = \rho(W_j)$, as else either W_i or W_j would be w-maximal. However then, W is only $w^\star$-maximal if either $W_i = \emptyset$ or $V_j = \emptyset$, leading to a contradiction.

Thus, the second case remains and we can assume we have a prefix of jobs $V \subsetneq J_i^a$. Note that for all $V \subsetneq J_i^a$ where V is a prefix of J_i^a we have $\rho(V) < \rho(J_i^a)$ since weights are only strictly increasing along each chain. Thus, no such V would ever be chosen as a $w^\star$-maximal in Algorithm 3. By Theorem 10, in an optimal schedule we would thus never schedule any such V, but only J_i^a. Thus, all jobs in J_i^a will be at some point picked as the next subset of jobs by Algorithm 3 to be scheduled to successive time slots. □

Now, consider a chain J_i, $1 \leq i \leq l$. For $\gamma \in M$, let $J_i^\gamma = \{j \in J_i \mid S^\star(j) = \gamma\}$ be the set of jobs of chain J_i which are scheduled to machine γ by the schedule $S^\star$, as defined in the proof of Lemma 11. Consider two machines $\alpha, \beta \in M$. Let $U \subseteq J_i^\alpha$ be a subset of jobs and let j_l, j_u be the jobs which are minimal and maximal for U with respect to the order $\prec$, i.e., $j_l \preceq j \preceq j_u$ for all $j \in U$. U is called *α, β-consecutive* if there is no $k \in J_i^\beta$ such that $j_l \preceq k \preceq j_u$. U is called an *α, β-block* if U is α, β-consecutive and if there is no proper superset $V \supsetneq U$ such that V is α, β-consecutive.

By $B_{\alpha,\beta}(i)$ we denote the set of all α,β- and β,α-blocks. For notational convenience, we refer to the elements of $B_{\alpha,\beta}(i)$ as $\{\alpha,\beta\}$-blocks, i.e., an $\{\alpha,\beta\}$-block is either an α,β-block or a β,α-block.

Lemma 12. *We have $|B_{\alpha,\beta}(i)| \leq 3$ for all $\alpha,\beta \in M$ and J_i, $1 \leq i \leq l$. I.e., on each pair of machines α,β each chain is split into at most three $\{\alpha,\beta\}$-blocks.*

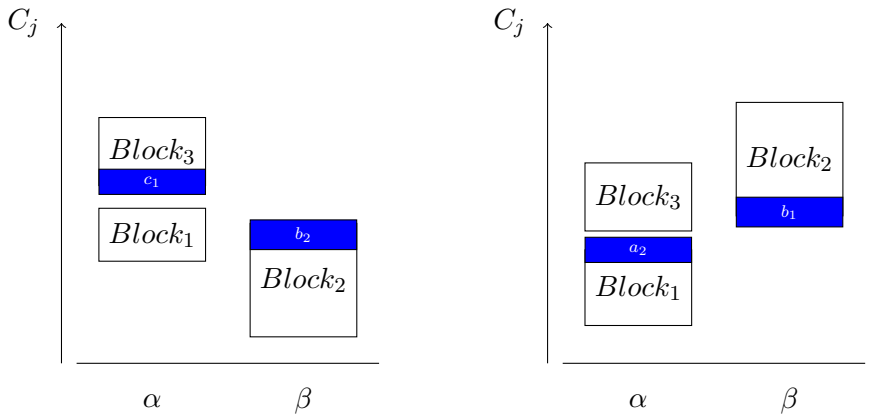
Proof. Fix some chain J_i, for some $1 \leq i \leq l$, and let $\alpha,\beta \in M$. Going forward, all jobs we consider will be from the chain J_i. To prove the statement, we will show that there is only a single possible arrangement of three $\{\alpha,\beta\}$-blocks on the two machines α,β. W.l.o.g., we assume that there are two α,β-blocks and one β,α-block on the two machines α,β. Given this unique possible arrangement of three $\{\alpha,\beta\}$-blocks, we show that no further $\{\alpha,\beta\}$-block can occur.

Assume that in $S^\star$, we have on the machines α,β an arrangement of two α,β-blocks and one β,α-block. In the following, for the statements to be well-defined, we consider the two α,β-blocks and the β,α-block which are scheduled to the earliest time slots on the machines α,β. We denote the α,β-block on α which is scheduled to earlier time slots in $S^\star$ as $Block_1$ and denote the later α,β-block on α as $Block_3$. Note that since $S^\star$ is a feasible schedule, we must have $j \prec k$ for all $j \in Block_1, k \in Block_3$. We denote the β,α-block on β as $Block_2$. Let a_1, a_2 be the minimal and maximal jobs of $Block_1$. Similarly, let b_1, b_2 be the minimal and maximal jobs of $Block_2$ and let c_1, c_2 be the minimal and maximal jobs of $Block_3$. We must have $a_2 \prec b_1 \prec b_2 \prec c_1$ for there to be two α,β-blocks and one β,α-block. Thus, overall we have $a_1 \preceq a_2 \prec b_1 \preceq b_2 \prec c_1 \preceq c_2$. Therefore, since all chains are ordered backwardly by assumption, we have $w_{a_1} \leq w_{a_2} < w_{b_1} \leq w_{b_2} < w_{c_1} \leq w_{c_2}$. Equality holds if two jobs are equal, i.e., if a block consists of a single job.

We now consider the completion times in $S^\star$ of the jobs mentioned above, namely $C_{a_1}, C_{a_2}, C_{b_1}, C_{b_2}, C_{c_1}$ and C_{c_2}. Trivially, we have $C_{a_1} \leq C_{a_2} < C_{c_1} \leq C_{c_2}$ and $C_{b_1} \leq C_{b_2}$. Note that given Lemma 11, all jobs of the same chain on the same machine are scheduled to consecutive time slots and thus we have $C_{c_1} = C_{a_2} + 1$. In the following, we consider different relations of the completion times of the jobs to each other. To show that there is a unique configuration of three $\{\alpha,\beta\}$-blocks in terms of the completion times, we proceed by assuming that some relation of the completion times relative to each other occurs in an optimal schedule. We then show how to rearrange (a subsets of) jobs such that we still have a feasible schedule but such that the sum of weighted completion times strictly decreases, leading to a contradiction with $S^\star$ being optimal. Intuitively, we move jobs with higher weights to earlier time slots while moving jobs with lighter weights to later time slots. We do this for every possible arrangement of completion times until only one option remains. Thus, we end up with only a single possible arrangement of three $\{\alpha,\beta\}$-blocks in terms of the completion times of their extremal jobs.

First, we show that $C_{c_1} \leq C_{b_2}$ and that $C_{b_1} \leq C_{a_2}$. Intuitively, this means that all three blocks must overlap in terms of the time slots they are scheduled to.

First, assume for a contradiction that $C_{b_2} < C_{c_1}$. Consider the schedule S' which we obtain from $S^\star$ by swapping jobs c_1 and b_2. See the left panel of Figure 2.2 for a visualization of the jobs to be swapped. The right panel of Figure 2.2 shows which jobs should be swapped in the case that we assume $C_{b_1} > C_{a_2}$, which we will discuss later. We show the arrangement of jobs before the swap and mark the jobs to be swapped in blue.



Assuming $C_{b_2} < C_{c_1}$, showing which swap leads to a contradiction

Assuming $C_{b_1} > C_{a_2}$, showing which swap leads to a contradiction

Figure 2.2

Consider the jobs of the blocks $Block_1, Block_2, Block_3$ on machine α under S'. In increasing order of completion times, we first have the jobs of $Block_1$, then job b_2 and then the jobs of $Block_3$ except c_1. On β, we have first the jobs of $Block_2$ except b_2 and then job c_1. All other jobs remain unchanged. Thus, the order of completion times in S' agrees with $\prec$ and S' is a feasible schedule.

Let C'_j denote the completion time of job $j \in J$ in S', while C_j denotes the completion time in $S^\star$. We have

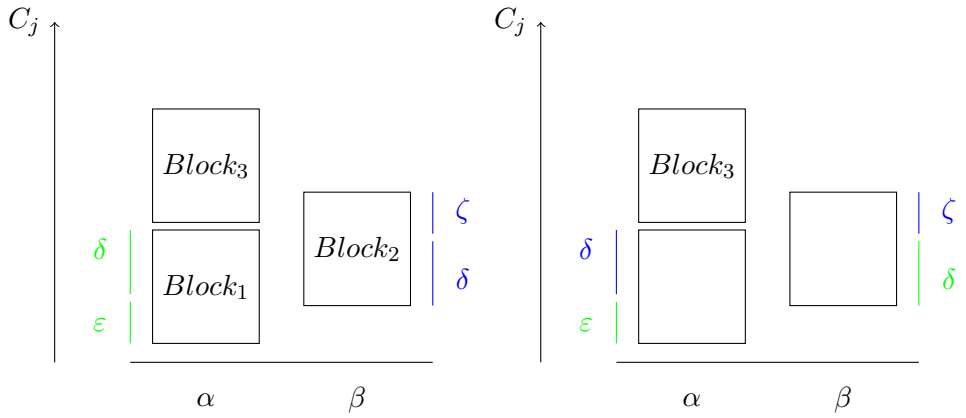
$$\begin{aligned} \textstyle\sum_{j\in J} w_j C'_j &= \textstyle\sum_{j\in J} w_j C_j - (w_{c_1} C_{c_1} + w_{b_2} C_{b_2}) + (w_{c_1} C_{b_2} + w_{b_2} C_{c_1}) \\ &= \textstyle\sum_{j\in J} w_j C_j + (w_{c_1} - w_{b_2})\,(C_{b_2} - C_{c_1}) \end{aligned}$$

Since $w_{b_2} < w_{c_1}$ and $C_{b_2} < C_{c_1}$ by assumption, we have $\sum_{j\in J} w_j C'_j < \sum_{j\in J} w_j C_j$, contradicting the optimality of $S^\star$. Thus, $C_{c_1} \leq C_{b_2}$ must hold. By a similar argument, we must have $C_{b_1} \leq C_{a_2}$, as else we could obtain a new schedule by swapping jobs a_2 and b_1.

Next, we show that $C_{b_1} \leq C_{a_1}$ and $C_{c_2} \leq C_{b_2}$ must hold. It will thus follow that we have $C_{b_1} \leq C_{a_1} \leq C_{a_2} < C_{c_1} \leq C_{c_2} \leq C_{b_2}$. Intuitively, this means that among the three blocks, $Block_2$ starts being processed first and is finished last.

To show this, first assume for a contradiction that $C_{a_1} < C_{b_1}$. Let $\delta := \min\{|Block_1|, |Block_2|\}$, $\varepsilon := |Block_1| - \delta$ and $\zeta := |Block_2| - \delta$. Note that if

$|Block_2| \geq |Block_1|$, then $|Block_1| = \delta$ and $\varepsilon = 0$. Else, if $|Block_2| < |Block_1|$, then $|Block_2| = \delta$ and $\zeta = 0$. In either case, $|Block_1| = \delta + \varepsilon$ and $|Block_2| = \delta + \zeta$. Consider the schedule S' obtained from $S^\star$ by swapping the last δ jobs of $Block_1$ with the first δ jobs of $Block_2$. I.e., depending on the cardinality of $Block_1$ and $Block_2$, we either swap the entire $Block_1$ with the first $\delta = |Block_1|$ jobs of $Block_2$ (if $|Block_2| \geq |Block_1|$) or we swap the entire $Block_2$ with the last $\delta = |Block_2|$ jobs of $Block_1$ (if $|Block_2| < |Block_1|$). See Figure 2.3 for a visualization. In this figure, the green and blue bars next to the blocks indicate how the jobs should be swapped.



$Block_1, Block_2, Block_3$ in the schedule $S^\star$

The jobs of the original $Block_1, Block_2, Block_3$ in the schedule S'

Figure 2.3

Again, consider the jobs of $Block_1, Block_2, Block_3$ on machine α under S'. In order of completion time, we first have the first ε jobs of $Block_1$, then the first δ jobs of $Block_2$ and afterwards all jobs of $Block_3$. On β, we first have the last δ jobs of $Block_1$ and then the last ζ jobs of $Block_2$. All other jobs remain unchanged. Thus, the order of completion times in S' agrees with $\prec$ and therefore S' is a feasible schedule.

Let C'_j denote the completion time of job $j \in J$ in S', while C_j denotes the completion time in $S^\star$. By $a^1, a^2, ..., a^\delta = a_2$ we refer to the last δ jobs of $Block_1$ in $S^\star$. By $b_1 = b^1, b^2, ..., b^\delta$ we refer to the first δ jobs of $Block_2$ in $S^\star$. Note that we have $w_{a^1} < ... < w_{a^\delta} < w_{b_1} < ... < w_{b^\delta}$ since the chain is ordered backwardly. Thus, in particular we have $w_{a^i} < w_{b^i}$ for all $1 \leq i \leq \delta$.

By assumption, we have $C_{a_1} < C_{b_1}$. Also, as shown before, we have $C_{a_2} < C_{c_1} \leq C_{b_2}$. First consider the case that $|Block_2| \geq |Block_1|$. Then, we have $\delta = |Block_1|$ and $\varepsilon = 0$. Therefore, $a_1 = a^1$ and thus $C_{a^1} = C_{a_1} < C_{b_1} = C_{b^1}$. Else, we have $|Block_2| < |Block_1|$ and $\delta = |Block_2|$ and $\zeta = 0$. Therefore, $b_2 = b^\delta$ and thus $C_{a^\delta} = C_{a_2} < C_{b_2} = C_{b^\delta}$. Since by assumption all jobs have unit processing times,

we have in both cases that $C_{a^i} < C_{b^i}$ for all $1 \leq 1 \leq \delta$. It follows that

$$\begin{aligned} \sum_{j \in J} w_j C'_j &= \sum_{j \in J} w_j C_j - \sum_{i=1}^{\delta} (w_{a^i} C_{a^i} + w_{b^i} C_{b^i}) + \sum_{i=1}^{\delta} (w_{a^i} C_{b^i} + w_{b^i} C_{a^i}) \\ &= \sum_{j \in J} w_j C_j + \sum_{i=1}^{\delta} (w_{b^i} - w_{a^i}) (C_{a^i} - C_{b^i}) \end{aligned}$$

Thus, we have $(w_{b^i} - w_{a^i}) (C_{a^i} - C_{b^i}) < 0$ for all $1 \leq i \leq \delta$ and $\sum_{j \in J} w_j C'_j \leq \sum_{j \in J} w_j C_j$. This contradicts $S^\star$ being optimal. It follows that $C_{b_1} \leq C_{a_1}$ must hold.

We obtain $C_{c_2} \leq C_{b_2}$ by a similar argument. We set $\delta := \min\{|Block_2|, |Block_3|\}$, $\varepsilon := |Block_3| - \delta$ and $\zeta := |Block_2| - \delta$. We obtain S' from $S^\star$ by swapping the first δ jobs of $Block_3$ with the last δ jobs of $Block_2$. From this point onwards the argument proceeds similarly as above.

Therefore, we must have $C_{b_1} \leq C_{a_1} \leq C_{a_2} < C_{c_1} \leq C_{c_2} \leq C_{b_2}$. We give a visualization of this case in Figure 2.4

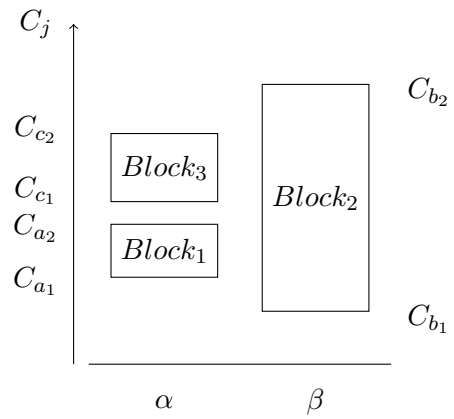


Figure 2.4: Two α, β-blocks and one β, α-block on two machines α, β

Given the unique arrangement of three $\{\alpha, \beta\}$-blocks in terms of the completion times, we consider if there can be four $\{\alpha, \beta\}$-blocks on two machines α, β in $S^\star$. Assume this is the case. W.l.o.g., we assume that the earliest three such $\{\alpha, \beta\}$-blocks consist of two α, β-blocks and one β, α-block. We denote the minimal and maximal jobs of these blocks as before. We have seen that $C_{b_1} \leq C_{a_1} \leq C_{a_2} < C_{c_1} \leq C_{c_2} \leq C_{b_2}$ must hold. The fourth $\{\alpha, \beta\}$-block must be a β, α-block, as else there would not be four $\{\alpha, \beta\}$-blocks. We denote this β, α-block as $Block_4$. Let d_1, d_2 be the minimal and maximal job of $Block_4$. We have $c_2 \prec d_1$ (for there to be four $\{\alpha, \beta\}$-blocks) and thus $w_{c_2} < w_{d_1}$ (since chains are ordered backwardly). Note that by Lemma 11 we have $C_{d_1} = C_{b_2} + 1$. From $C_{c_2} \leq C_{b_2}$, as shown above, it thus follows that $C_{d_1} > C_{c_2}$. In this setting, we could switch c_2 with d_1. This would again lead to a feasible schedule S', as the order of the completion times in S' on both machines α, β agrees with the order $\prec$. We again have $C_{c_2} < C_{d_1}$ and

$w_{c_2} < w_{d_1}$. Therefore, by the same analytical steps as above, we have that S' has a smaller sum of weighted completion times than $S^\star$, leading to a contradiction. Thus there cannot be a fourth $\{\alpha, \beta\}$-block in any optimal solution and the lemma follows. □

Consider $U \subseteq J_i$ and let j_l, j_u be the minimal and maximal jobs of U with respect to the order $\prec$, i.e., $j_l \preceq j \preceq j_u$ for all $j \in U$. We call U *consecutive* (without reference to any pair of machines) if $S^\star(j) = \alpha$ for all $j \in U$ for some fixed machine $\alpha \in M$ and if there is no $k \in J_i$ such that $j_l \preceq k \preceq j_u$ and $S^\star(k) \neq \alpha$. U is called a *block* (without reference to any pair of machines) if U is consecutive and if there is no proper superset $V \supsetneq U$ such that V is consecutive. By $B(i)$ we denote the set of all blocks of J_i across all machines.

To clarify the relationship between the two sets of definitions for blocks and α, β-blocks, consider the following. Let $B \subseteq J_i^\alpha$ be an α, β-block and let $B' \subseteq J_i^\alpha$ be a block. Let j_l, j_u be the minimal and maximal job of B and let j'_l, j'_u be the minimal and maximal job of B'. We can think of B as a maximal subset of jobs of chain J_i scheduled to a single machine α such that we have $\forall k \in J_i^\beta : \neg(j_l \preceq k \preceq j_u)$. We can think of B' as a maximal subset of jobs of chain J_i scheduled to a single machine α such that we have $\forall \beta \in M \backslash \{\alpha\} \forall k \in J_i^\beta : \neg(j'_l \preceq k \preceq j'_u)$ Thus intuitively, a block is a "refinement" of an α, β-block. Stated differently, for an α, β-block we consider a single machine β, while for a block, we consider all other machines $\beta \neq \alpha$. Thus, each block is an α, β-block as well, for all $\beta \neq \alpha$, where α is the machine the jobs of the block are scheduled to. An α, β-block on the other hand can be comprised of several smaller blocks.

Given this, we can now state the main structural lemma:

Lemma 13. *Let J_i, $1 \leq i \leq l$ be some chain and let $\alpha \in M$ be the machine the minimal job of J_i is scheduled to. Let b be the number of blocks of chain J_i scheduled to machine α.*

Then, we can partition $M = \bigcup_{a=0}^{b+1} M_a$ into disjoint subsets such that $M_a \neq \emptyset$ for $0 \leq a < b$ and such that for each pair of jobs $j, k \in J_i$ with $j \in J_i^{\bar{u}}, k \in J_i^{\bar{v}}, \bar{u} \in M_u, \bar{v} \in M_v$ and $1 \leq \bar{u} < \bar{v} \leq m$ we have $j \prec k$.

Proof. Given the rather technical statement of the lemma, let us start by providing some intuition and insights into the statement. We want to show that we can partition the set of machines such that jobs which appear later in the chain are scheduled to machines which are part of a subset with higher index in the partition. Let B_1 be the block which contains the minimal job of chain J_i. Let α be the machine B_1 is scheduled to. We define $M_0 := \{\alpha\}$. Let $M_{b+1} := \{a \in M \mid J_i^a = \emptyset\}$ be the set of machines such that no job of chain J_i is scheduled to them by $S^\star$. Note that we can have that $M_{b+1} = \emptyset$. Let $\bar{M} := M \backslash (M_0 \cup M_{b+1})$.

Given this, we want to show that we can partition $\bar{M} = \bigcup_{a=1}^{b} M_a$ into disjoint subsets, such that $M_a \neq \emptyset$ for $1 \leq a < b$ and such that the jobs of chain J_i assigned to some machine $\bar{v} \in M_v$ appear later in the chain than all jobs assigned to some

machine $\bar{u} \in M_u$ with $1 \leq u < v \leq b$. Note that we can write here $1 \leq u < v \leq b$ instead of $1 \leq \bar{u} < \bar{v} \leq m$, since we can always re-label the machines appropriately.

We consider two cases: First, we assume that $b = 1$, thus $J_i^\alpha = B_1$ and all jobs of chain J_i scheduled to machine α are part of the same block B_1. In this case, we have $\bar{M} = M_1$, and the statement of the lemma follows trivially.

Second, we assume that $b > 1$. Let $B, B' \in B(i)$ be two blocks. We say B *is earlier than* B', written $B <_{B(i)} B'$, if for all $j \in B$ and $k \in B'$ we have $j \prec k$. Given the definition of blocks above and given that $S^\star$ is a feasible schedule, for each pair of blocks $B \neq B'$ we either must have $B <_{B(i)} B'$ or $B' <_{B(i)} B$. Additionally, we have $\neg\left(B <_{B(i)} B\right)$, $B <_{B(i)} B' \Rightarrow \neg\left(B' <_{B(i)} B\right)$ and $\left(B <_{B(i)} B'\right) \wedge \left(B' <_{B(i)} B''\right) \Rightarrow \left(B <_{B(i)} B''\right)$ since $\prec$ is a total order on J_i. Thus, $<_{B(i)}$ defines a total order on $B(i)$. Starting from B_1, we now iteratively define $B_a \subseteq J_i^\alpha$ for $2 \leq a \leq b$ as the block on machine α scheduled immediately after block B_{a-1}, i.e., as the block such that $B_{a-1} <_{B(i)} B_a$ and such that for all blocks $B \subseteq J_i^\alpha$ with $B \neq B_\beta$, for $1 \leq \beta \leq a-1$, we have $B_a <_{B(i)} B$. We define $\mathcal{B}_a := \left\{B \in B(i) \mid j \notin J_i^\alpha \text{ for all } j \in B, B_a <_{B(i)} B <_{B(i)} B_{a+1}\right\}$ for $1 \leq a \leq b$, i.e., $\mathcal{B}_a$ is the set of blocks between B_a and B_{a+1} scheduled to machines different from α. In the definition above, we refer to B_{b+1}, which is not defined. We do so for notational convenience and assume that $B <_{B(i)} B_{b+1}$ for all blocks $B \in B(i)$. For $1 \leq a < b$, we have $\mathcal{B}_a \neq \emptyset$, as else B_a and B_{a+1} wouldn't be two different blocks. We can have $\mathcal{B}_b = \emptyset$, if the maximal job of J_i is scheduled to machine α. Else, $\mathcal{B}_b \neq \emptyset$. Note that $\bigcup_{a=1}^{b} (B_a \cup \mathcal{B}_a) = B(i)$ is a partition of $B(i)$ given that $<_{B(i)}$ is a total order.

For $1 \leq a \leq b$ we define $M_a := \{\gamma \in M \mid \exists B \in \mathcal{B}_a \exists j \in B : j \in J_i^\gamma\}$ as the set of machines jobs from $\mathcal{B}_a$ are scheduled to. Note that $\bar{M} = \bigcup_{a=1}^{b} M_a$. We will see that this is a partition of $\bar{M}$ which satisfies the statement of the lemma. By definition of the $\mathcal{B}_a$ and since $<_{B(i)}$ is a total order on $B(i)$, for all pairs of jobs $j, k \in J_i$ with $j \in J_i^{\bar{u}}, k \in J_i^{\bar{v}}, \bar{u} \in M_u, \bar{v} \in M_v$ and $1 \leq u < v \leq b$ we have $j \prec k$. For $1 \leq a < b$ we have $M_a \neq \emptyset$, as $\mathcal{B}_a \neq \emptyset$, as seen above. Lastly, consider $M_a, M_{a'}$ with $1 \leq a < a' \leq b$ (assuming $M_b \neq \emptyset$, else with $1 \leq a < a' < b$). Assume for a contradiction that there exists a machine $\beta \in M_a \cap M_{a'}$. Consider the block B_1, some block $B \in \mathcal{B}_a$, some block $B' \in \mathcal{B}_{a'}$ ans last the block $B_{a'}$. Together, B_1, B, B' and $B_{a'}$ imply the existence of four $\{\alpha, \beta\}$-blocks on the machines α, β. This however contradicts Lemma 12. Thus, we must have $M_a \cap M_{a'} = \emptyset$ and the statement holds. □

Corollary 14. *We have $|B(i)| \leq 2m - 1$ for J_i, $1 \leq i \leq l$. I.e., each chain is split into at most $2m - 1$ blocks across all machines.*

Proof. Let $S^\star$ be the optimal schedule for our current instance of PO Scheduling. Let $M_0 \subseteq M$ be a subset of machines and denote $M_1 := M \backslash M_0$. We define $J_i := \bigcup_{m \in M_i} J^i$ for $i = 0, 1$. Let S_i be the schedule for the jobs J_i on the machines M_i obtained from deleting the machines M_{1-i} and jobs J_{1-i} from $S^\star$. Note that S_i must be an optimal schedule for PO Scheduling on the instance with jobs J_i and

machines M_i. Else, w.l.o.g., we consider $i = 0$. Assume there is a schedule $\bar{S}$ with a strictly lower sum of weighted completion times on the instance with J_0 and M_0 than S_0. Then, we can combine the schedules $\bar{S}$ and S_1 into a feasible schedule for the instance with jobs J and machines M since jobs on different machines don't interact with each other in terms of feasibility. The combined schedule has a strictly lower sum of weighted completion times than $S^\star$, leading to a contradiction.

To prove the statement of the lemma, we proceed by induction over m. For $m = 1$, the statement is trivial. For the induction step, let $\alpha \in M$ be the machine the minimal job of J_i is scheduled to. Let b be the number of blocks of chain J_i scheduled to machine α and let $M = \bigcup_{a=0}^{b+1} M_a$ be the partition obtained from Lemma 13. We use all notation as introduced in the proof of Lemma 13.

If $b = 1$, we have $M_1 = \bar{M}$. The statement follows by the induction hypothesis, as there are at most $2|M_1| - 1 = 2(m-1) - 1 = 2m - 3$ blocks on M_1. We apply the induction hypothesis to the instance of PO Scheduling where we delete machine α and all jobs J^α scheduled to α in $S^\star$ as described in the introduction of the proof. Thus, in total we obtain $2m - 3 + 1 = 2m - 2$ blocks and the statement follows.

If $b > 1$, we consider $\bar{M} = \bigcup_{a=1}^{b} M_a$ as obtained from Lemma 13. Let $m_a = |M_a|$ for $1 \leq a \leq b$. By Lemma 13, for $1 \leq a < b$, we have $M_a \neq \emptyset$. Note that we can have $M_b = \emptyset$ and thus $m_b = 0$. By the induction hypothesis, we have at most $2m_a - 1$ blocks on M_a. We apply the induction hypothesis to the instance of PO Scheduling where we split our current instance into M_a and $M \backslash M_a$ as described in the introduction of the proof. As $\bigcup_{a=1}^{b} M_a$ is a partition of $\bar{M}$, we have $\sum_{a=1}^{b} m_a = |\bar{M}| \leq m - 1$. Adding up all individual terms yields us $|B(i)| \leq b + \sum_{a=1}^{b-1} (2m_a - 1) + 0 \leq b + 2(m-1) - (b-1) = 2m - 1$. □

Thus, we see that in an optimal solution $S^\star$ to the special case of PO Scheduling discussed in this section, the chains are structured in a certain restricted way as described above.

2.3.3 An IP Formulation for the Multiple-Chains Case

Given Corollary 14, we can try to formulate an IP to encapsulate the special case of PO Scheduling we currently consider. The goal would be to use such an IP to find an LP-based approximation algorithm. However, we will see that we encounter significant roadblocks with this approach, which leaves us to abandon it for now. While we give an IP 2.2 below, this fails to fully capture our special case while also taking advantage of the structure of optimal schedules explored in Corollary 14.

Let $(J, \prec), M$ be an instance of PO Scheduling. As before, we assume that $p_j = 1$ for all $j \in J$ and that $\prec$ consists of l disjoint backwardly ordered chains. We define a set of time slots $T := [n]$, where $n := |J|$. We have binary variables x_{jt} for all $j \in J$ and all $t \in T$ where we interpret $x_{jt} = 1$ as job j being scheduled to some machine at time slot $t \in T$. Additionally, we have variables $u_j \in [2m-1]$ for all $j \in J$. Let $j \in J_i$ for some $1 \leq i \leq l$. $u_j = \alpha$ indicates job j is part of the α^{th} block of chain J_i. Last, we have binary variables z_{jk} for all $j, k \in J$ where we interpret $z_{jk} = 1$

as jobs j and k being scheduled to different machines. We consider the following time-indexed IP:

$$\begin{array}{lllll} \min & \sum_{j\in J}\sum_{t\in T} w_j t x_{jt} & & & \\ \text{s.t.} & \sum_{t\in T} x_{jt} & = & 1 & \forall j \in J \\ & \sum_{j\in J} x_{jt} & \leq & m & \forall t \in T \\ & \sum_{t'<t} x_{jt'} + z_{jk} & \geq & \sum_{t'\leq t} x_{kt'} & \forall t \in T, \forall j \prec k \\ & u_k & \geq & u_j + z_{jk} & \forall j \prec k \\ & u_j & \in & [2m-1] & \forall j \in J \\ & z_{jk} & \in & \{0,1\} & \forall j,k \in J \\ & x_{jt} & \in & \{0,1\} & \forall j \in J, \forall t \in T \end{array} \tag{2.2}$$

Note that the IP 2.2 takes inspiration from time-indexed IP-formulations for precedence-constrained scheduling [6]. The first constraint expresses that each job must be scheduled at some time slot. The second constraint states that at each time slot at most m jobs can be scheduled since we have m machines. The third constraints encodes that the order of completion times agrees with the order $\prec$ for jobs scheduled to the same machine. If jobs are scheduled to different machines, the constraint is trivially satisfied. In the fourth constraint, we state that every time we change machines for a chain, we must increase the suitable block counting variable u. The fifth constraint bounds the size of the block counting variables u based on Corollary 14.

In the linear relaxation of 2.2 we relax the integrality constraints to $1 \leq u_j \leq 2m-1$ for all $j \in J$, $0 \leq z_{jk} \leq 1$ for all $j,k \in J$ and $0 \leq x_{jt} \leq 1$ for all $j \in J$ and for all $t \in T$.

Note that the IP 2.2 is inspired by the IP 2.1 for TO Scheduling in terms of the u-variables. In the IP 2.1, we can interpret the u-variables as machine indices, based on Lemma 5 about Smith-monotone schedules. In the IP 2.2, we cannot interpret the u-variables as machine indices directly, as we can only bound them from above using Corollary 14 about how often a single chain switches machines. Also, different chains might disagree on the ordering of the machines, i.e., which machines are visited in which order. Thus, for the IP 2.2 we cannot interpret the u-variables as machine indices directly.

Additionally, note that in case that we have chains of length at most $2m-1$, the fifth constraint does not provide any additional structure to the solution. However, recall that Algorithm 2 allows us to approximate cases where all chains are short. This justifies some optimism that if we can find an LP-based approximation algorithm based on IP 2.2 (or some other IP which makes use of the structure found in Corollary 14) we can also find some way to incorporate Algorithm 2 to deal with short chains as well, where the IP 2.2 does not provide any additional structure.

To be able to derive an LP-based approximation algorithm from the IP 2.2, we must have that the cost of an optimal solution to the linear relaxation cannot have cost which is significantly lower than the cost of an optimal integral solution. To analyze LP-based approximation algorithms, we usually bound the cost of the

integral optimum by the cost of the fractional optimum. When both costs are too far apart, we cannot use the cost of the fractional optimum in a suitable way. For the IP 2.2 to be suitable to be used in an LP-based approximation algorithm, the property above must hold, in particular for the case of a single chain, i.e., TO Scheduling, as well. We have seen in Section 2.2 that the objective value of the fractional optimal solution of the linear relaxation and the integer optimal solution of the IP 2.1 coincide. In the IP 2.1, we bound the u-variables from above by m, whereas in the IP 2.2 we bound the u-variables from above by $2m - 1$. We will shortly consider an instance for TO Scheduling, where even changing the number of machines by 1 leads to an arbitrarily large change of the value of the fractional optimum for the linear relaxation of the IP 2.1. Given the structural similarities of the IPs 2.1 and 2.2 and given that the IP 2.2 heavily uses the $2m - 1$ bound (instead of the m bound from the IP 2.1), we are pessimistic that the IP 2.2 can be successfully used to design an LP-based approximation algorithm for TO Scheduling or therefore for PO Scheduling. Thus, we won't further pursue this approach for now. Potentially, utilizing the IP 2.2 it might be possible to derive a *bi-criteria result*, i.e., a result where we find an integral solution with cost within a constant factor of the fractional optimum, but which uses e.g., twice as many machines. We, however, leave this question open for now.

We now consider the instance of TO Scheduling mentioned above. More formally, we show we cannot bound the increase in the optimal sum of weighted completion times value of the linear relaxation of the IP 2.1 when removing a single machine in TO Scheduling. While not contained in [3], the following instance is due to Sitters (one of the authors of [3]) and was conveyed in personal communication with his coauthors. We here still assume unit processing times.

Lemma 15 (Sitters)**.** *Consider the TO Scheduling IP-formulation 2.1 on some instance $(J, <), M$ and let OPT be the optimal value of said instance to the linear relaxation to 2.1. Let $(J, <), M'$ be a new instance of TO Scheduling where we obtain M' from M by removing a single machine. Let OPT' be the optimal value of new instance $(J, <), M'$ to the linear relaxation to 2.1. Then, $\frac{OPT'}{OPT}$ can be arbitrarily large.*

Proof. Consider the following instance of TO Scheduling $(J, <), M$, where $|J| = n$ and $|M| = m$. The jobs are split into m different "batches" $(D_i)_{i \in [m]}$. Batches with higher index contain fewer jobs but have a higher total weight than batches with lower index. In particular for an arbitrary but fixed $k \in \mathbb{N}_{>0}$, batch D_i consists of k^{m-i+1} jobs, each with weight $\frac{1}{k^{2(m-i+1)}}$. Thus, the total weight of each batch is $\frac{1}{k^{m-i+1}}$. There are in total $n = k^m + ... + k^2 + k$ jobs. Within each batch, the jobs are totally ordered by $<$. Jobs in batches with lower index proceed jobs in batches with higher index with regard to $<$.

An optimal schedule for $(J, <), M$ simply puts every batch i on its own machine i. The cost of this solution is $OPT = \sum_{i=1}^{m} k^{m-i+1} \frac{k^{m-i+1}+1}{2} \frac{1}{k^{2(m-i+1)}} \approx \frac{1}{2}m$. Now suppose we decrease the number of machines from m to $m - 1$. By the pigeonhole

principle, there must be at least one machine that contains a $\frac{1}{m-1}$ fraction of the jobs from two different batches. We now consider such a machine. We call the fraction of jobs coming from the earlier (respectively later) of the two batches D (respectively D'), and suppose D' comes from a batch with k^i jobs. Thus, $|D'| \geq \frac{1}{m-1}k^i$ and $|D| \geq \frac{1}{m-1}k^{i+1}$. The weight of each job in D' is $\frac{1}{k^{2i}}$. Since the jobs in D are scheduled before those in D', the completion times of jobs in D' is at least $|D| \geq \frac{1}{m-1}k^{i+1}$. Therefore, the total cost of jobs in D' is at least $\frac{1}{k^{2i}}\frac{1}{(m-1)^2}k^i k^{i+1} = \Omega(\frac{k}{m^2})$. This is already a lower bound for OPT', the optimal value of the new instance $(J, <), M'$. Since we can choose k arbitrarily large, this completes the argument. □

Thus, as stated before, it seems unlikely that we can use the IP 2.2 to design an LP-based approximation algorithm for PO Scheduling or even for the multiple-chains case. Therefore, we don't further pursue this approach for now.

2.4 The Configuration IP

In this section, we consider a quite different idea (compared to the approach presented in Section 2.3) on how to design a constant-factor approximation algorithm for PO Scheduling. The approach is LP-based as well, i.e., we give an IP formulation for PO Scheduling and discuss solving and rounding the linear relaxation. We reduce solving the linear relaxation to solving a different scheduling problem, *Scheduling with Rejection* and show the current state of progress with this approach. While we do not solve the problem here, there are potential jumping-off points for further research.

2.4.1 The Configuration LP and its Dual

Let $(J, \prec)$, M be an instance of PO Scheduling. For each subset of jobs $C \subseteq J$ we denote by $S^\star_C$ an optimal schedule of C on a single machine, where the order of the completion times agrees with the partial order $\prec$. By $w(S^\star_C)$ we denote the cost of said schedule. More generally, if S is some schedule, we denote by $w(S)$ the sum of the weighted completion times of this schedule. Recall that on a single machine, PO Scheduling is equivalent to precedence-constrained scheduling on a single machine[7]. As mentioned before, precedence-constrained scheduling is NP-hard, even for the case of a single machine. Note that for every given $C \subseteq J$, we can find a 2-approximation for $w(S^\star_C)$ [22]. For now, let us set this issue aside and treat $w(S^\star_C)$ as some known values.

We start by considering the *Configuration IP*. For each $C \subseteq J$ we have a binary variable z_C, where $z_C = 1$ indicates that there is some machine in the final schedule that has exactly the jobs in C scheduled to it.

[7] $1|prec|\sum_j w_j C_j$ in the three-field notation by [20].

$$\begin{array}{lllll} \min & \sum_{C\subseteq J} w(S_C^\star) z_C & & & \\ \text{s.t.} & \sum_{C\subseteq J} z_C & \leq & m & \\ & \sum_{C\subseteq J: j\in C} z_c & = & 1 & \forall j \in J \\ & z_C & \in & \{0,1\} & \forall C \subseteq J \end{array} \tag{2.3}$$

The constraints of the IP 2.3 imply that we want to find at most $m = |M|$ subsets of jobs (each scheduled to its own machine) such that each job is contained in exactly one subset of jobs. Consider the linear relaxation of the IP 2.3, where we replace the integrality constraint by $0 \leq z_C \leq 1$ for all $C \subseteq J$. We call this linear relaxation the *Configuration LP*. Note that both the IP and its linear relaxation have exponentially many variables, as there are exponentially many subsets $C \subseteq J$. Given the linear relaxation, we have to find solutions to two main questions to arrive at a constant-factor approximation algorithm for PO Scheduling based on the IP 2.3:

- Solving an LP can be done in polynomial time in terms of amount of variables and constraints the LP has. Given that the Configuration LP has exponentially many variables in terms of the input size, can we find an optimal fractional solution in polynomial time in terms of the input size?
- Given an optimal fractional solution to the Configuration LP, how can we obtain a feasible integral solution of comparable cost? I.e., this is the problem of rounding an optimal LP solution.

We will focus on the first question for the rest of this section and leave the second question open for now. As noted, in the Configuration LP we have exponentially many variables and polynomially many constraints. To deal with the exponential number of variables, we consider the dual LP of the Configuration LP (which we will call *primal* from now). LP Duality is a fundamental concept in the field of linear optimization. An introduction on LP Duality and the most important results can be found in Schrijver [42]. In particular, if we find an optimal solution to the dual problem, we can from it compute an optimal solution to the primal problem (using strong duality and complementary slackness [42]). In the dual problem (or *dual* for short), we will have polynomially many variables and exponentially many constraints. More specifically, in the dual we have one variable λ associated with the first constraint of the primal and one variable y_j for each $j \in J$, related to each instance of the second type of constraints in the primal. We obtain the dual LP 2.4:

$$\begin{array}{lllll} \max & -m\lambda + \sum_{j\in J} y_j & & & \\ \text{s.t.} & -\lambda + \sum_{j\in C} y_j & \leq & w(S_C^\star) & \forall C \subseteq J \\ & \lambda & \geq & 0 & \end{array} \tag{2.4}$$

Consider some class of inequalities $Ax \leq b$. The problem of checking if a point $\bar{x}$ satisfies $A\bar{x} \leq b$ is called the *separation problem* (for this class of inequalities). It is known (by using Khachiyan's Ellipsoid Method [26] and by using a version of

binary search [42] to find the optimal value) that solving the separation problem in polynomial time enables optimizing over the same inequalities in polynomial time, i.e., when we can solve the separation problem for $Ax \leq b$ in polynomial time, we can find $x^\star \in \max\{c(x) \mid Ax \leq b\}$, for an objective vector c of suitable dimension, in polynomial time.

Consider the separation problem for the dual LP 2.4 For a point (λ, y) we can trivially check $\lambda \geq 0$ in constant time. We have exponentially many inequalities of the form $-\lambda + \sum_{j \in C} y_j \leq w(S^\star_C)$ for which we have to check if they are all satisfied. Checking if all such inequalities are satisfied is equivalent to checking $\max_{C \subseteq J} \left(-\lambda + \sum_{j \in C} y_j - w(S^\star_C)\right) \leq 0$. The question remains if we can find said maximum in polynomial time. If so, as explained above, we can find the dual optimum in polynomial time and thus the primal optimum as well.

2.4.2 The Dual Separation Problem

Let us consider the dual separation problem for a given point (λ, y) in more detail. Ideally, we would like to check if $\max_{C \subseteq J} \left(-\lambda + \sum_{j \in C} y_j - w(S^\star_C)\right) \leq 0$ in polynomial time. We can reformulate the separation problem as follows:

$$\begin{aligned}
& \max_{C \subseteq J} \left(-\lambda + \textstyle\sum_{j \in C} y_j - w(S^\star_C)\right) && \leq 0 \\
\Leftrightarrow\ & \max_{C \subseteq J} \left(\textstyle\sum_{j \in C} y_j - w(S^\star_C) - \sum_{j \in J} y_j\right) && \leq \lambda - \textstyle\sum_{j \in J} y_j \\
\Leftrightarrow\ & \min_{C \subseteq J} \left(w(S^\star_C) + \textstyle\sum_{j \in J} y_j - \sum_{j \in C} y_j\right) && \geq \textstyle\sum_{j \in J} y_j - \lambda \\
\Leftrightarrow\ & \min_{C \subseteq J} \left(w(S^\star_C) + \textstyle\sum_{j \notin C} y_j\right) && \geq \textstyle\sum_{j \in J} y_j - \lambda
\end{aligned}$$

Thus, the separation problem can be expressed as a minimization problem over all possible subsets of jobs, where we need to check if the optimal value of our objective function takes at least the value $\sum_{j \in J} y_j - \lambda$.

Note that we can interpret this minimization problem again as a scheduling problem with the following setting: We are given a partially ordered set of jobs $(J, \prec)$, the same set as in the original input for PO Scheduling, and a single machine $M = \{1\}$. Recall that on a single machine, PO Scheduling is equivalent to precedence-constrained scheduling. For each job $j \in J$, we have weights w_j, processing times p_j and rejection costs y_j which we obtain from the point (λ, y). We can either schedule a job, leading to some weighted completion time, or pay y_j to reject the job, meaning that the job won't contribute any completion time. A schedule is valid if the order of the completion times of the scheduled jobs agrees with the partial order $\prec$. The goal is to find a valid schedule that minimizes the sum of weighted completion times plus the sum of rejection payments incurred. This problem is know as scheduling with rejection and precedence constraints on one single machine[8]. The

[8] Extending the 3-field notation and following the notation of [47, 13], the problem can be denoted as $1|rej, prec| \sum_S w_j C_j + \sum_{\bar{S}} y_j$, where $S \subseteq J$ is the set of jobs to be scheduled, $\bar{S} = J \backslash S$ and rej denotes the option for rejection.

wider class of scheduling problems where jobs can be rejected is known as *scheduling with rejection* and is a well-studied class of scheduling problems in its own right, see e.g., [47, 13]. We abbreviate scheduling with rejection with precedence constraints on one single machine in this section to *"Scheduling with Rejection"* for the sake of brevity when the context is clear.

The question remains how to solve the separation problem in polynomial time. Recall from the beginning of this section that finding $w(S^\star_C)$ is already NP-hard. Thus, we cannot hope to solve the separation problem optimally in polynomial time. Given this, we consider finding a suitable approximation algorithm for the dual separation problem.

Let (λ, y) be a point for which we want to solve dual the separation problem, i.e., Scheduling with Rejection, where we interpret y_j as rejection cost for each $j \in J$. To approximate the separation problem in a suitable manner, we consider finding an approximation algorithm with a special property. Let α be some fixed constant which will be our desired approximation factor. We define $y'_j := \frac{y_j}{\alpha}$ for all $j \in J$. We consider an algorithm that in polynomial time returns a subset of jobs $\bar{C}$ and a valid schedule $\bar{S}_{\bar{C}}$ (in terms of precedence constraints from $\prec$) for all jobs in $\bar{C}$ to a single machine, with cost $w(\bar{S}_{\bar{C}})$, such that

$$w(\bar{S}_{\bar{C}}) + \alpha \sum_{j \notin \bar{C}} y'_j \leq \alpha \min_{C \subseteq J} \left\{ w(S^\star_C) + \sum_{j \notin C} y'_j \right\}. \tag{2.5}$$

Such an algorithm is called a *Lagrange-Multiplier preserving* (LMP) approximation algorithm for Scheduling with Rejection with rejection cost y'_j for each $j \in J$. LMP approximation algorithms are used in designing approximation algorithms for different problems [18, 30] and can intuitively be understood as approximation algorithms where we "localize" the approximation error to only some component of the cost function. In our case, we want to "localize" the approximation factor to the part of the cost stemming from the weighted completion times. We describe in the rest of this section how such an LMP algorithm helps us to find an approximate solution to the Configuration LP.

Let $C^\star$ be such that $w(S^\star_{C^\star}) + \sum_{j \notin C^\star} y_j = \min_{C \subseteq J} \left\{ w(S^\star_C) + \sum_{j \notin C} y_j \right\}$. Alternatively, as seen above, $-\lambda + \sum_{j \in C^\star} y_j - w(S^\star_{C^\star}) = \max_{C \subseteq J} \left(-\lambda + \sum_{j \in C} y_j - w(S^\star_C) \right)$. It can be easily checked that the condition 2.5 is equivalent to

$$w(\bar{S}_{\bar{C}}) + \sum_{j \notin \bar{C}} y_j \leq \alpha w(S^\star_{C^\star}) + \sum_{j \notin C^\star} y_j. \tag{2.6}$$

Given that 2.5 and 2.6 are equivalent, we work with the later condition going forward and thus we can use the rejection costs y_j for each $j \in J$ (obtained from (λ, y)) directly instead of the $y'_j = \frac{y_j}{\alpha}$ defined earlier.

We discuss finding such an LMP approximation algorithm for the dual separation problem in the next Subsection 2.7. For now, we consider how having access to an LMP approximating algorithm would help us in designing an approximation algorithm for PO Scheduling. Recall that (λ, y) is a point for which we want to solve the dual separation problem and let $\bar{C}, \bar{S}_{\bar{C}}, w(\bar{S}_{\bar{C}})$ be the output of the LMP approximation algorithm for the current point. We can check if $-\lambda + \sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \leq 0$ or not. If $-\lambda + \sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) > 0$, we have found a new violated inequality related to $\bar{C}, \bar{S}_{\bar{C}}, w(\bar{S}_{\bar{C}})$ and can thus say that (λ, y) is not feasible. If however $-\lambda + \sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \leq 0$, we have the following:

$$\begin{aligned} 0 &\geq -\lambda + \textstyle\sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \\ &= -\lambda - \textstyle\sum_{j\notin\bar{C}} y_j - w(\bar{S}_{\bar{C}}) + \sum_{j\in J} y_j \\ &\geq -\lambda - \textstyle\sum_{j\notin C^\star} y_j - \alpha w(S^\star_{C^\star}) + \sum_{j\in J} y_j \\ &= -\lambda + \textstyle\sum_{j\in C^\star} y_j - \alpha w(S^\star_{C^\star}) \end{aligned}$$

Thus, from $-\lambda+\sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \leq 0$ it follows $-\lambda+\sum_{j\in C^\star} y_j - \alpha w(S^\star_{C^\star}) \leq 0$. Note that by definition of $C^\star$ we have $-\lambda + \sum_{j\in C^\star} y_j - w(S^\star_{C^\star}) \geq -\lambda + \sum_{j\in C} y_j - w(S^\star_C)$ for all $C \subseteq J$. Putting these two facts together, from $-\lambda + \sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \leq 0$ it follows $-\lambda + \sum_{j\in C} y_j - \alpha w(S^\star_C) \leq 0$ for all $C \subseteq J$. In other words, having $-\lambda+\sum_{j\in\bar{C}} y_j - w(\bar{S}_{\bar{C}}) \leq 0$ gives witness that (λ, y) approximately satisfies the dual separation problem, i.e., (λ, y) satisfies all inequalities if we scale up the right-hand side by a factor α.

Given what we have seen above, we can now state how to use an LMP algorithm as described above to obtain an approximately optimal solution to the Configuration LP. We can apply Khachiyan's Ellipsoid Method [26] to figure out in polynomial time if (an α-scaled version of) the LP 2.4 is feasible based on solving the separation problem. By doing so, if the α-scaled version of the problem is indeed feasible, we obtain in polynomial time a solution $(\lambda^\star, y^\star)$ that has objective value at least the optimal value of 2.4 and which satisfies all inequalities of 2.4 if the right-hand side is scaled up by a factor α[9].

Let s be the number of times the LMP algorithm is called until finding the point $(\lambda^\star, y^\star)$ as described above. As mentioned earlier, s can be bounded by a polynomial in terms of the input size. By $\bar{C}_i$ we denote the subset of jobs that was returned the i^{th} time the LMP approximation algorithm was called for $1 \leq i \leq s$. Let $\mathcal{S} := \{\bar{C}_i \mid 1 \leq i < s\}$. Let $\bar{S}_{\bar{C}_i}$ be the schedule returned by the LMP approximation algorithm along with $\bar{C}_i$. Consider the LP 2.7:

$$\begin{array}{lllll} \max & -m\lambda + \sum_{j\in J} y_j & & & \\ \text{s.t.} & -\lambda + \sum_{j\in C} y_j & \leq & w(\bar{S}_{\bar{C}}) & \forall \bar{C} \in \mathcal{S} \\ & \lambda & \geq & 0 & \end{array} \tag{2.7}$$

[9] We basically apply the well-known methodology to solving a linear problem to optimality by reducing it to a feasibility problem and applying Khachiyan's Ellipsoid Method. For further details, see Schrijver [42].

Note that $(\lambda^\star, y^\star)$ is optimal for the LP 2.7 by design of which constraints we include. Equivalently, $(\lambda^\star, y^\star)$ has objective value at least the objective value of the optimum to the LP 2.4 and satisfies all inequalities of 2.4 if the right-hand side is scaled up by a factor α. Consider the dual of the LP 2.7:

$$\begin{array}{lllll} \min & \sum_{\bar{C}\in\mathcal{S}} w(\bar{S}_{\bar{C}}) z_{\bar{C}} & & & \\ \text{s.t.} & \sum_{\bar{C}\in\mathcal{S}} z_{\bar{C}} & \leq & m & \\ & \sum_{\bar{C}\in\mathcal{S}: j\in\bar{C}} z_{\bar{C}} & = & 1 & \forall j \in J \\ 0 \leq & z_{\bar{C}} & \leq & 1 & \forall \bar{C} \in \mathcal{S} \end{array} \tag{2.8}$$

The LP 2.8 has the same types of constraints as the Configuration LP but only $|\mathcal{S}| = s-1$ many variables. Thus, we can find an optimal solution to 2.8 in polynomial time in terms of the input size. Let z be such an optimal solution. We can extend z to a feasible solution $\bar{z}$ for the Configuration LP by setting $\bar{z}_C = 0$ for all $C \notin \mathcal{S}$. By the theorem of strong duality [42] the optimal solution z to the LP 2.8 has the same optimal value as $(\lambda^\star, y^\star)$ in the LP 2.7. Thus, since $\bar{z}$ is obtained from z since $\bar{z}$ and $(\lambda^\star, y^\star)$ are corresponding primal and dual optimal solutions, and since $(\lambda^\star, y^\star)$ has objective value at least the objective value of the optimum to the LP 2.4 and satisfies all inequalities of 2.4 if the right-hand side is scaled up by a factor α, $\bar{z}$ can be understood as a solution for the Configuration LP where each term in the objective function was scaled up by at most a factor of α. Overall, this is equivalent to saying that $\bar{z}$ is an α-approximation to the optimal solution to the linear relaxation of 2.3.

Overall, if we can find an LMP approximation algorithm for Scheduling with Rejection, we can obtain an α-approximation for the fractional optimum solution, which would be a strong first step to design an approximation algorithm for PO Scheduling based on the Configuration IP 2.3. Thus, in the next part of this section, we consider finding a LMP constant-factor approximation algorithm for Scheduling with Rejection.

2.4.3 Approximating Scheduling with Rejection

We have seen in the previous section, that to make progress on PO Scheduling, we are interested in finding a constant-factor LMP approximation algorithm for Scheduling with Rejection.

As mentioned, scheduling with rejection in general is a commonly studied subclass of scheduling problems and is considered in various flavors [13]. Related to our discussions, Engels, Karger, Kolliopoulos, Sengupta, Uma and Wein [13] have shown that there is a constant-factor (non-LMP) approximation algorithm for scheduling with rejection, release-times and precedence constraints on a single machine[10]. As mentioned, this approximation algorithm is not LMP. We, however, still want to give

[10] $1|rej, r_j, prec|\sum_S w_j C_j + \sum_{\bar{S}} y_j$ in the extended 3-field notation. In scheduling with release-times, each job $j \in J$ is only available to be scheduled from a known release-time $r_j \geq 0$ onwards. We obtain scheduling without release-times as a special case by assuming $r_j = 0$ for all $j \in J$.

an overview of the result here, as it might provide a jumping-off point for further research.

The approach of Engels, Karger, Kolliopoulos, Sengupta, Uma and Wein [13] is quite general and can be used to extend approximation algorithms for certain assignment problems to approximation algorithms for the same assignment problem with rejection. The notation used here follows [13]. Consider a problem $\mathcal{P}$, where we want to assign each object in some set $\mathcal{O}_1$ to objects in some set $\mathcal{O}_2$. Each $i \in \mathcal{O}_1$ should be assigned to s_i objects in $\mathcal{O}_2$, where s_i is some natural number. To each $j \in \mathcal{O}_2$ at most κ_j items from $\mathcal{O}_1$ can be assigned to, where κ_j is some natural number. $M \subseteq \mathcal{O}_1 \times \mathcal{O}_2$ denotes a set of forbidden assignments. There might be further additional constraints. The goal is to minimize some objective function, while satisfying all constraints. The authors model the problem as an IP, using binary decision variables x_{ij} for all $i \in \mathcal{O}_1$ and all $j \in \mathcal{O}_2$, where $x_{ij} = 1$ is interpreted as i being assigned to j. Furthermore, the authors introduce variables y_l for $l \in [|\mathcal{O}_1| + |\mathcal{O}_2|]$, one for each object, to model potential further constraints and costs. Let c, d be non-negative cost-vectors of suitable dimensions, A be a constraint-matrix of suitable dimensions and let b be a right-hand side vector of suitable dimensions. $A \begin{bmatrix} x & y \end{bmatrix}^\top \leq b$ captures any additional constraints that might be present. Thus, the authors consider the following IP-formulation of $\mathcal{P}$:

$$\begin{array}{lllll} \min & c^\top x + d^\top y & & & \\ \text{s.t.} & \sum_{j \in \mathcal{O}_2} x_{ij} & = & s_i & \forall i \in \mathcal{O}_1 \\ & \sum_{i \in \mathcal{O}_1} x_{ij} & \leq & \kappa_j & \forall j \in \mathcal{O}_2 \\ & A \begin{bmatrix} x & y \end{bmatrix}^\top & \leq & b & \\ & x_{ij} & = & 0 & \forall (i,j) \in M \\ & x_{ij} & \in & \{0,1\} & \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 \end{array} \tag{2.9}$$

Consider the linear relaxation of 2.9, where we replace the integrality constraints $x_{ij} \in \{0,1\}$ for all $i \in \mathcal{O}_1, j \in \mathcal{O}_2$ by $x_{ij} \geq 0$ for all $i \in \mathcal{O}_1, j \in \mathcal{O}_2$. The authors call this relaxation LP1. Now assume there exists an approximation algorithm $\mathcal{A}$ with approximation factor ρ for the problem $\mathcal{P}$ that operates by rounding an optimal solution to LP1 to a feasible solution to the original problem, where the cost of the rounded solution is at most ρ times the cost of the optimal solution to LP1. Note that in the original problem $\mathcal{P}$, no rejection is present. The authors show that, given certain conditions, the approximation algorithm $\mathcal{P}$ can be extended to an approximation algorithm for $\mathcal{P}$ where we allow for rejection, i.e., it is possible to reject elements from $\mathcal{O}_1$ instead of assigning them.

To do so, the authors first consider a further relaxation of LP1, which they refer to as LP2. Here, $0 < \beta < 1$ is some constant. We now allow that an object in $\mathcal{O}_1$ is assigned only at least βs_i times, instead of being assigned exactly s_i times. Thus,

we obtain:

$$\begin{array}{lllll} \min & c^\top x + d^\top y & & & \\ \text{s.t.} & \sum_{j\in\mathcal{O}_2} x_{ij} & \geq & \beta s_i & \forall i \in \mathcal{O}_1 \\ & \sum_{i\in\mathcal{O}_1} x_{ij} & \leq & \kappa_j & \forall j \in \mathcal{O}_2 \\ & A\begin{bmatrix} x & y \end{bmatrix}^\top & \leq & b & \\ & x_{ij} & = & 0 & \forall (i,j) \in M \\ & x_{ij} & \geq & 0 & \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 \end{array} \tag{2.10}$$

Given LP2, we can state the first condition introduced by the authors.

Condition 16 ([13])**.** *Let q be a feasible solution to LP2 of value $v(q)$. There exists a polynomial-time algorithm to convert q into a feasible solution to LP1 of value at most $f(\beta)v(q)$, where f is some function of β.*

Given this first condition, we can now introduce rejection to the original problem $\mathcal{P}$. We refer to the problem with rejection as $\mathcal{P}_R$. The authors model $\mathcal{P}_R$ as an IP, by introducing new variables z_i for each $i \in \mathcal{O}_1$, where we interpret $z_i = 1$ as i getting rejected. Rejecting i has costs g_i associated to it. We obtain the following IP:

$$\begin{array}{lllll} \min & c^\top x + d^\top y + g^\top z & & & \\ \text{s.t.} & \frac{1}{s_i}\left(\sum_{j\in\mathcal{O}_2} x_{ij}\right) + z_i & = & 1 & \forall i \in \mathcal{O}_1 \\ & \sum_{i\in\mathcal{O}_1} x_{ij} & \leq & \kappa_j & \forall j \in \mathcal{O}_2 \\ & A\begin{bmatrix} x & y \end{bmatrix}^\top & \leq & b & \\ & x_{ij} & = & 0 & \forall (i,j) \in M \\ & x_{ij} & \in & \{0,1\} & \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 \\ & z_i & \in & \{0,1\} & \forall i \in \mathcal{O}_1 \end{array} \tag{2.11}$$

Similarly as before, the authors introduce LP1_R, where we replace the integrality constraints by $x_{ij} \geq 0$ for all $i \in \mathcal{O}_1, j \in \mathcal{O}_2$ and $z_i \geq 0$ for all $i \in \mathcal{O}_1$.

Let (x, y, z) be a feasible solution to LP1_R and let $0 < \beta < 1$ be a positive constant. The authors define $S_\beta := \left\{ i \in \mathcal{O}_1 \mid \sum_{j\in\mathcal{O}_2} x_{ij} \geq \beta s_i \right\}$ as the set of objects which are fractionally assigned in total to βs_i objects in $\mathcal{O}_2$. The authors refer by $x(S_\beta)$ and $y(S_\beta)$ to the vectors x and y restricted to the entries in S_β. Given these definitions, we can state the second condition.

Condition 17 ([13])**.** *Let (x, y, z) be a feasible solution to LP1$_R$. Then, for any β, we have that $(x(S_\beta), y(S_\beta))$ is a feasible solution to LP2, where we consider only S_β as elements to be assigned.*

Thus, the authors can give an Algorithm 4 for $\mathcal{P}_R$ as follows, assuming that Conditions 16 and 17 as satisfied.

Note that in step 2 of the Algorithm 4 the rejection part of the cost only increases by a factor of at most $\frac{1}{1-\beta}$ since at least a $1 - \beta$ fraction of the jobs was already rejected by design.

Therefore, the authors obtain the following theorem:

Algorithm 4: $\max\{f(\beta)\rho, \frac{1}{1-\beta}\}$-Approximation Algorithm for $\mathcal{P}_R$

Input : An instance of the problem $\mathcal{P}_R$, an approximation algorithm $\mathcal{A}$ for $\mathcal{P}$ with approximation factor ρ, a value $0 < \beta < 1$ and a function f of β.

Output : A feasible solution to $\mathcal{P}_R$ with cost at most $\max\{f(\beta)\rho, \frac{1}{1-\beta}\}$ times the optimum.

1 Compute an optimal solution to LP1_R (x, y, z).

2 Define $S_\beta := \left\{ i \in \mathcal{O}_1 \mid \sum_{j \in \mathcal{O}_2} x_{ij} \geq \beta s_i \right\}$ and let $\bar{S} := \mathcal{O}_1 \backslash S$ be the set of rejected objects.

3 Note that by Condition 17, $(x(S_\beta), y(S_\beta))$ is a feasible solution to LP2. Convert that solution into a feasible solution $(\bar{x}, \bar{y})$ to LP1, using Condition 16.

4 Apply the approximation algorithm $\mathcal{A}$ to $(\bar{x}, \bar{y})$ and obtain a solution $(x^\star, y^\star)$.

5 Return $(x^\star, y^\star)$.

Theorem 18 ([13])**.** *Let $\mathcal{P}$ be a problem satisfying Conditions 16 and 17. If there is an approximation algorithm for $\mathcal{P}$ such that the cost of the solution obtained by the approximation algorithm is at most ρ times the cost of an optimal solution to LP1, then Algorithm 4 is a $\max\{f(\beta)\rho, \frac{1}{1-\beta}\}$-approximation algorithm for $\mathcal{P}_R$.*

Given this theorem, the authors consider an IP-formulation for scheduling with rejection, release-times and precedence constraints on a single machine. They consider the a time-indexed IP. Here, $T := [\max_{j \in J}\{r_j\} + \sum_{j \in J} p_j]$ is a set of time slots jobs can be scheduled to.

$$
\begin{array}{lllll}
\min & \sum_{j \in J} w_j C_j & & & \\
\text{s.t.} & \sum_{j \in J} x_{jt} & \leq & 1 & \forall t \in T \\
& \sum_{t \in T \backslash [0, r_j]} x_{jt} & = & p_j & \forall j \in J \\
& \frac{1}{p_j} \sum_{r_j \leq t' < t} x_{jt'} & \geq & \frac{1}{p_k} \sum_{r_k \leq t' \leq t} x_{kt'} & \forall j \prec k, j, k \in J, \forall t \in T \\
& C_j & = & \frac{p_j}{2} + \frac{1}{p_j} \sum_{t \in T} x_{jt} \left(t + \frac{1}{2}\right) & \forall j \in J \\
& x_{jt} & = & 0 & \forall j \in J, t < r_j \\
& x_{jt} & \in & \{0, 1\} & \forall j \in J, t \in T
\end{array}
\tag{2.12}
$$

The authors show that the IP 2.12 along with the relaxations introduced above satisfy Conditions 16 and 17. Thus, by Theorem 18, they show that any approximation algorithm for scheduling with release-times and precedence constraints on a single machine, which is based on the IP 2.12 can be extended to an approximation algorithm for scheduling with rejection, release-times and precedence constraints on a single machine. For this result we have $f(\beta) = \frac{1}{\beta^2}$ and $\beta = \frac{1}{2}\left(\sqrt{\rho\,(\rho + 4)} - \rho\right)$. The approximation algorithm they use for scheduling with release-times and prece-

dence constraints on a single machine is due to Schulz and Skutella [44] with an approximation factor of $(e + \varepsilon)$ for $\varepsilon > 0$.

Lemma 19 ([13])**.** *For every $\varepsilon > 0$, there is a $(4.5 + \varepsilon)$-approximation algorithm for scheduling with rejection, release-times and precedence constraints on a single machine.*

There are more recent approximation algorithms for scheduling with release-times and precedence constraints on a single machine. The currently best known approximation factor is $(2 + \varepsilon)$ for $\varepsilon > 0$ [49]. However, their authors use different IP-formulations, meaning the result presented above does not immediately carry over to these different IP-formulations without some further work.

By setting all release-times to zero, we immediately obtain a constant-factor non-LMP approximation algorithm for PO Scheduling with rejection on a single machine. However, it is not immediately clear if the arguments presented above due to Engels, Karger, Kolliopoulos, Sengupta, Uma and Wein [13] can be extended or modified to obtain an LMP approximation algorithms for Scheduling with Rejection. We leave this potential path of further research open for now.

Given the discussion on a non-LMP approximation algorithm for Scheduling with Rejection, let us now consider finding an LMP approximation algorithm for the problem. While we do not find such an algorithm here, we will find a more direct way of how a LMP approximation algorithm for Scheduling with Rejection leads to an approximate solution to the Configuration LP than was shown earlier in Section 2.4.2.

Recall the notation from the previous subsection. In Scheduling with Rejection, we are given a partially ordered set of jobs $(J, \prec)$, where $n := |J|$ with weights w_j, processing times p_j and rejection costs y_j for each job $j \in J$. We can either schedule a job or reject it, but then incur the rejection cost. The goal is to find a subset of jobs to be scheduled to a single machine and a feasible schedule (where the order of the scheduled jobs on the machine agrees with the partial order $\prec$) with the lowest sum of the weighted completion times for the scheduled jobs plus the rejection costs for the rejected jobs. As a first step, we focus on the case of unit processing times, i.e., $p_j = 1$ for all $j \in J$.

Consider the following IP 2.13 for Scheduling with Rejection. It is a time-indexed IP and quite similar to the IP formulation 2.12 used for the non-LMP approximation algorithm. Let $T := [n]$ be a set of time slots. We have binary variables x_{jt} for all $j \in J$ and $t \in T$, where $x_{jt} = 1$ indicates that job j is scheduled to time slot t. Additionally, we have binary variables z_{jt} for all $j \in J$ and $t \in T$, where $z_{jt} = 1$ indicates that job j is rejected at time slot t. While the specific time slot at which a job is rejected is not important in terms of the goal of Scheduling with Rejection, we will see below that these variables form a convenient extended formulation of the problem.

$$
\begin{array}{lllll}
\min & \sum_{j\in J}\sum_{t\in T}(w_j t x_{jt} + y_j z_{jt}) & & & \\
\text{s.t.} & \sum_{j\in J} x_{jt} & \leq & 1 & \forall t \in T \\
& \sum_{t\in T}(x_{jt} + z_{jt}) & = & 1 & \forall j \in J \\
& \sum_{t'<t}\left(x_{jt'} + z_{jt'}\right) + z_{jt} & \geq & \sum_{t'\leq t}(x_{kt'} + z_{kt'}) & \forall j \prec k, j, k \in J, \forall t \in T \\
& x_{jt} & \in & \{0,1\} & \forall j \in J, t \in T \\
& z_{jt} & \in & \{0,1\} & \forall j \in J, t \in T
\end{array}
\tag{2.13}
$$

We want at each time slot that at most one job is scheduled to the time slot (first constraint). Each job must be either scheduled or rejected (second constraint). Additionally, if job k with $j \prec k$ is scheduled or rejected at some time slot, job j must have been scheduled or rejected at some previous time slot, or j is rejected at the same time slot k is scheduled to (third constraint). Consider the linear relaxation of 2.13, where we replace the integrality constraints by $0 \leq x_{jt}, z_{jt} \leq 1$ for all $j \in J$ and all $t \in T$.

We will see that having access to an LMP approximation algorithm for Scheduling with Rejection that operates by rounding an optimal solution to the linear relaxation of 2.13 leads to a more direct way of finding an approximate solution to the Configuration LP as we have shown in the previous section 2.4.2. To do so, let $\delta := \frac{1}{m}$. Recall that $m = |M|$, where M is the machine set in the original instance of PO Scheduling we consider. Assume we have an instance of Scheduling with Rejection where in an optimal fractional solution (x, z) we have $\sum_{t\in T} z_{jt} = 1 - \delta$ for all $j \in J$, i.e., we want to "almost reject" all jobs. Assume now that we have a way to round the fractionally optimal solution (x, z) to a distribution over schedules that respect precedence constraints and where each schedule rejects some jobs and schedules the rest on a single machine, and each job is scheduled with probability precisely $\frac{1}{m}$. If we scale up this distribution by a factor of m, it is precisely a solution to the Configuration LP for PO scheduling on m machines. Thus, with such a rounding procedure in place, we could for such an instance find an approximate solution to the Configuration LP much more directly.

However, we can always achieve the first condition by adding the constraint $\sum_{t\in T} z_{jt} = 1 - \delta$ for all $j \in J$ to the IP 2.13 and to its linear relaxation. We refer to the linear relaxation of 2.13 with the additional constraint as $\mathrm{LP}_{\mathrm{add}}$. Assume now we have a (randomized) LMP approximation algorithm with approximation factor α, where α is some positive constant, for Scheduling with Rejection that takes an optimal solution to $\mathrm{LP}_{\mathrm{add}}$ and rounds it to a feasible schedule. This implies that the integrality gap of $\mathrm{LP}_{\mathrm{add}}$ is at most α. By Yao's principle [59], this is equivalent to the following: for every solution (x, z) to $\mathrm{LP}_{\mathrm{add}}$ there is a distribution over integral solutions to $\mathrm{LP}_{\mathrm{add}}$ such that probability that a job j is included in a schedule is at most $\alpha \sum_{t\in T} x_{jt}$. Thus, the probability that a job j is scheduled is at most $\alpha \sum_{t\in T} x_{jt} = \alpha\delta = \alpha\frac{1}{m}$. Therefore, besides the constant factor α, we are exactly in the setting described above.

Thus, having access to an LMP approximation algorithm for Scheduling with Rejection would provide a more direct way to get an approximately optimal solution to the Configuration LP, as we have just seen. We leave the question of finding such an LMP approximation algorithm open for now.

Chapter 3

The Weighted Matroid Augmentation Problem

In this chapter, we consider the *"Weighted Matroid Augmentation Problem"*, which we will abbreviate as *WMAP*. We describe the background and the motivation of the problem in Section 3.1. Here, we also state the main Conjecture 20 of this chapter. In Section 3.2, we give the main definitions needed throughout the chapter and explain in some more detail why we state Conjecture 20 in the specific way we do. In Section 3.3 we show that Conjecture 20 holds for the two important base-cases of graphic and cographic matroids. In the last two Sections 3.4 and 3.5, we present two possible ways of approaching Conjecture 20 based on iterative rounding (Section 3.4) and via Seymour's decomposition theorem (Section 3.5). We explain what the techniques are based on, give preliminary results, show limitations encountered so far and point towards further possible approaches on trying to prove Conjecture 20.

3.1 Problem Description and Motivation

The *Tree Augmentation Problem* (TAP) is a fundamental and heavily studied [16, 39, 10, 33, 1] problem in the field of augmentation algorithms. Let $G = (V, E)$ be an undirected graph with a spanning tree $T \subseteq E$. We call the edges in T *tree-edges* and the edges in $L := E \backslash T$ *links*. The goal is to find a minimal set $S \subseteq L$ such that $(V, T \cup S)$ is 2-edge-connected. Given a cost-function $c : L \to \mathbb{R}_{\geq 0}$, we can ask for finding the cost-minimal set of links S, such that the resulting graph is 2-edge-connected. This version of the problem in referred to as the *Weighted Tree Augmentation Problem* (WTAP).

Already TAP, even on trees of diameter 4, was shown to be APX-hard by Kortsarz, Krauthgamer and Lee [31]. Thus, TAP and WTAP are studied under the point of view of approximation algorithms. For TAP, the currently best known approximation factor is 1.393 by Cecchetto, Traub and Zenklusen [5], improving from a factor of 1.458 [21] and previously a factor of 1.5 [32, 8, 14]. For a long time, the

best known approximation factor for WTAP has been 2, which is due to Frederickson and Jájá [16]. This factor was recently brought down by Traub and Zenklusen, first to $(1 + \ln 2 + \varepsilon) < (1.7 + \varepsilon)$ and then to $(1.5 + \varepsilon)$ [51, 52].

(W)TAP can be understood as a special case of the *(Weighted) Connectivity Augmentation Problem*, (W)CAP. Here, we are given an undirected graph $G = (V, E)$ and a k-edge-connected subgraph $H \subseteq E$ for some $k \in \mathbb{N}_{>0}$. Again, we call the edges in $L := E \backslash H$ *links* and are given a cost-function $c : L \to \mathbb{R}_{\geq 0}$. The goal is to find a cost-minimal (or cardinality-minimal for the case of CAP) subset of links $S \subseteq L$ such that $(V, H \cup S)$ is $k + 1$-edge-connected. In the case of $k = 1$, we can contract all 2-edge-connected components of H and thus attain a spanning T on a smaller graph, bringing us back to the setting of (W)TAP. Thus, (W)CAP is also APX-hard. In the case that k is odd, it has been known that CAP reduces to the case $k = 1$, i.e., to TAP [11]. For CAP, there has been recent progress improving the approximation factor from 1.91 [4] to 1.393 by Cecchetto, Traub and Zenklusen [5], the same result as mentioned above for TAP, thus unifying the progress on both problems. For WCAP, Traub and Zenklusen [53] gave a $(1.5 + \varepsilon)$-approximation, improving from the previous factor of 2 [17, 23].

Given the recent results, especially for WTAP, we want to consider a generalized version of the problem, based on matroids. We give a short overview of matroids in Section 3.2. We consider the following problem: Let $M = (E, \mathcal{I})$ be a matroid with a distinguished basis B and a cost-function $c : E \to \mathbb{R}_{\geq 0}$. For some $s \in E \backslash B$, let $C(s, B)$ be the unique fundamental cycle for s and B. The goal is to find a cost-minimal subset $S \subseteq E \backslash B$ such that the set $\{C(s, B) \mid s \in S\}$ covers B, i.e., for all $b \in B$ there is some $s \in S$ such that $b \in C(s, B)$. We call this problem *"Weighted Matroid Augmentation Problem"* (WMAP). As mentioned, we will discuss matroids and the related concepts in Section 3.2. In that section, we also re-consider the problem statement for WMAP and see that it is a generalization of WTAP. We state the following open conjecture:

Conjecture 20. *There is a 2-approximation algorithm for WMAP in the case that the matroid is regular.*

In the next Section 3.2, we define regular matroids and justify why we focus on them in Conjecture 20.

Equivalently, we can reformulateMatroidConjecture as follows:

Conjecture 21. *There is a 2-approximation algorithm for the set-covering problem in case that the constraint matrix is the support of a totally unimodular matrix.*

We will introduce the set-covering problem in Section 3.2.2 and totally unimodular matrices in Section 3.2.3. In Section 3.4.1 we discuss why both statements of the conjecture are equivalent.

3.2 An Introduction to Matroids

A deep overview of matroids and matroid theory can be found in a textbook by Oxley [40]. Schrijver [42] also gives a short introduction to the topic. For the sake of being self-contained, we will give a brief overview of some of the key definitions and results which are needed for in this chapter. The results mentioned in this section are quite well known and can all be found again in greater detail in [40].

3.2.1 Basic Definitions

Let E be a finite set, called the *ground set*, and let $\mathcal{I} \subseteq \mathcal{P}(E)$ be a family of subsets of E. We call $\mathcal{I}$ the *independent sets* and say a set $I \in \mathcal{I}$ is *independent*. $M = (E, \mathcal{I})$ is a *matroid* if M satisfies the following axioms:

1. The empty set is independent:
   $\emptyset \in \mathcal{I}$
2. Every subset of an independent set is independent:
   $\forall A \subseteq B \subseteq E : B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$
3. For all independent sets A, B, where A has smaller cardinality than B, there is an element $b \in B \backslash A$ such that $A \cup \{b\}$ is independent:
   $\forall A, B \in \mathcal{I} : |A| < |B| \Rightarrow (\exists b \in B \backslash A : A \cup \{b\} \in \mathcal{I})$

Matroids are widely studied objects in various branches of Mathematics, such as Combinatorial Optimization and Combinatorics. They generalize the concept of linear independence in vector spaces. We will see this connection in more detail when we discuss representable matroids in Section 3.2.2.

A set $A \notin \mathcal{I}$ is called a *dependent* set. A minimal dependent set, i.e., a set where every proper subset is independent, is called a *circuit* (of M). As mentioned, subsets $A \subseteq E$ with $A \in \mathcal{I}$ are called independent. A maximal independent set, i.e., a set such that every proper superset is dependent, is called a *basis* (of M). From the third matroid axiom above, it follows that all bases of M have the same cardinality. Consider a fixed basis $B \in \mathcal{I}$ and some element $e \in E \backslash B$[1]. By definition, $B \cup \{e\} \notin \mathcal{I}$. Thus, there is a circuit contained in $B \cup \{e\}$. It can be shown that there is exactly one circuit contained in $B \cup \{e\}$. This unique circuit is called the *fundamental circuit* $C(e, B)$ related to e and B. Note that by the second axiom we must have $e \in C(e, B)$.

Let $M = (E, \mathcal{I})$ be a matroid. Define $\mathcal{I}^\star := \{I \subseteq E \mid E \backslash I \text{ contains a basis of } M\}$. It can be shown that $M^\star = (E, \mathcal{I}^\star)$ is again a matroid [40]. $M^\star$ is called the *dual matroid* of M. Dual matroids play an important role in matroid theory, which we can here only acknowledge in passing. For a more thorough discussion, see [40].

[1] Assuming $B \neq E$. If $B = E$, by the second axiom $\mathcal{I} = \mathcal{P}(E)$ and there are no dependent sets anyways.

One example of a matroid is the *graphic matroid*. Consider an undirected connected graph $G = (V, E)$. Let $\mathcal{I}(G) := \{T \subseteq E \mid (V, T) \text{ contains no cycles}\}$. It can be checked that $M(G) := (E, \mathcal{I}(G))$ is a matroid. Furthermore, the set of bases of $M(G)$ is formed by the set of spanning trees of G, i.e., the maximal cycle-free subsets of the edges. We call such matroids *graphic matroids*[2].

Given this knowledge, we can revisit WMAP and see that in the case of a graphic matroid we indeed obtain the problem of weighted tree augmentation. In other words, WMAP is a generalization of WTAP. In WMAP, we are given a matroid $M = (E, \mathcal{I})$, a distinguished basis B and a cost-function $c : E \backslash B \to \mathbb{R}_{\geq 0}$. If M is a graphic matroid, we have seen that B is a spanning tree $T \subseteq E$ of the graph $G = (V, E)$, where G is implicitly given via the edges E. Let $L := E \backslash T$ be called the links. The cost-function $c : L \to \mathbb{R}_{\geq 0}$ can be understood as assigning some cost to each link. For WTAP, the goal is to find a cost-minimal subset $S \subseteq L$ such that the $(V, T \cup S)$ is 2-edge-connected. For WMAP, the goal is to find a cost-minimal subset $S \subseteq E \backslash B$ such that the set $\{C(s, B) \mid s \in S\}$ covers B. For the case of the graphic matroid, consider some $\{u, v\} = l \in L$. Note that in T there is an unique path between u and v, which we denote by $P(u, v)$ or $P(l)$. The fundamental circuit $C(l, T)$ is the unique cycle in $(V, T \cup \{l\})$, i.e., $C(l, T) = \{l\} \cup P(l)$. Note that by adding l, all nodes adjacent to edges in $P(l)$ are now 2-edge-connected. In terms of WMAP, adding l covers all edges in $C(l, T) \backslash \{l\} = P(l) \subseteq T$. Therefore, we can understand the goal of WMAP for the case of the graphic matroid as finding a cost-minimal set $S \subseteq L$ such that $(V, T \cup S)$ is 2-edge-connected., which is exactly the set-up of WTAP.

3.2.2 Representable Matroids

Let $\mathcal{F}$ be a field and consider a finite-dimensional vector-space V over $\mathcal{F}$. Let E' be a finite (multi-)set of vectors in V and let $\mathcal{I}'$ be a set of linear independent subsets of E'. It can be easily checked that $M' = (E', \mathcal{I}')$ is a matroid.

Now let $M = (E, \mathcal{I})$ be an arbitrary matroid and let V again be a finite-dimensional vector-space over a field $\mathcal{F}$. Let $f : E \to V$ be a function from the ground set E to the vector-space V. f is called a *$\mathcal{F}$-representation* of M if for all $A \subseteq E$ we have that $A \in \mathcal{I}$ if and only if $f|_A$ is injective and the set of vectors in $f(A)$ are linearly independent. If, for a matroid M and a field $\mathcal{F}$, there exists a finite dimensional vector-space V such that there is a $\mathcal{F}$-representation, we say that M is *$\mathcal{F}$-representable*. Intuitively, M is $\mathcal{F}$-representable if there is a function f that maps the elements in E to vectors in some finite-dimensional vector-space V over $\mathcal{F}$ such that independent sets in $\mathcal{I}$ are mapped to linearly independent sets of vectors (where the elements in independent sets must be mapped one to one, to avoid trivial

[2]This includes matroids isomorphic to such matroids. However, we will not focus on the concept of matroid isomorphisms here and, moving forward, we implicitly assume that we always include isomorphic matroids in our definitions.

maps). Clearly, the type of matroid described at the beginning of this subsection is $\mathcal{F}$-representable. We say a matroid M is *representable* (or *linear*) if there exists a field $\mathcal{F}$ such that M is $\mathcal{F}$-representable. Note that there are some matroids that are $\mathcal{F}$-representable over some fields $\mathcal{F}$ but not over all fields. Furthermore, there are non-representable matroids, i.e., matroids that are not $\mathcal{F}$-representable over any field $\mathcal{F}$. A matroid is called *regular* if for every field $\mathcal{F}$, M is $\mathcal{F}$-representable. We discuss such matroids in greater detail later in this chapter.

Let us consider again the problem WMAP. The goal is to find a cost-minimal subset $S \subseteq E \backslash B$ such that the set $\{C(s, B) \mid s \in S\}$ covers B. From this formulation, it is clear that we are considering a special case of the set-covering problem. For the set-covering problem, we denote by n the size of the ground set, i.e., the set to be covered. It is known that no approximation algorithm can have a better approximation factor than $O(\log n)$, unless $\mathsf{P} = \mathsf{NP}$, as shown by Raz and Safra [41]. Thus, we should not expect a constant-factor approximation algorithm for the general set-covering problem.

For a finite set S, consider a finite family $\mathcal{A} := (A_j)_{j \in J}$ of subsets of S, i.e., $A_j \subseteq S$ for all $j \in J$. We call $T \subseteq S$ a *transversal* of $\mathcal{A} = (A_j)_{j \in J}$, if there exists a bijective function $f : J \to T$ such that $f(j) \in A_j$. We call $X \subseteq S$ a *partial transversal* of S if there is a subset $K \subseteq J$ such that X is a transversal of $(A_j)_{j \in K}$. Let $\mathcal{I}(\mathcal{A})$ be the set of partial transversals of $\mathcal{A}$. It can be shown that $M(\mathcal{A}) := (S, \mathcal{I}(\mathcal{A}))$ is a matroid [40]. Such matroids are called *transversal matroids* where the transversals of S are the bases of the matroid $M(\mathcal{A})$. Proving the statement above can be achieved by interpreting partial transversals as matchings in a bipartite graph as follows: Let $G = (V, E)$ be a graph with $V := S \cup J$ and $E := \{\{s, j\} \mid s \in A_j, j \in J\}$. Clearly, G is bipartite. It is not difficult to see that $X \subseteq S$ is a partial transversal if and only if there is a matching in G covering every node in X, i.e., if there is a subset $Y \subseteq E$ of the edges such that no two edges in Y share a node and that every node in X is incident to exactly one edge in Y. The full proof that $M(\mathcal{A}) := (S, \mathcal{I}(\mathcal{A}))$ is a matroid can be found in [40]. Importantly for our case, it can be shown that transversal matroids are representable [40].

Theorem 22. *There is no constant-factor approximation algorithm for WMAP already for the case of representable matroids unless* $\mathsf{P} = \mathsf{NP}$.

Proof. We show this statement by considering instances of transversal matroids, which are representable [40].

Consider a finite set $S = \{s_1, s_2, ..., s_b\}$ and a finite family of subsets $\mathcal{A} := (A_j)_{j \in J}$ with $A_j \subseteq S$ for all $j \in J$. Let $B := [b]$. Now, consider the ground set $S' := S \cup J$ with a finite family of subsets $\mathcal{A}' := (A'_i)_{i \in B}$, where $A'_i := \{s_i\} \cup \{j \in J \mid s_i \in A_j\}$ for all $i \in B$. Let $M(\mathcal{A}')$ be the related transversal matroid over the ground set S'.

For $M(\mathcal{A}')$, consider the graph $G' = (V', E')$ as described above. According to the definition from earlier, we obtain $V' = S \cup J \cup B$ and $E' = \{\{s_k, k\} \mid k \in B\} \cup \{\{j, k\} \mid s_k \in A_j, k \in B\}$.

Note that $S \subseteq S'$ is a transversal of $\mathcal{A}'$. Thus, $S \subseteq S'$ is a basis of $M(\mathcal{A}')$. Consider the fundamental circuit $C(j, S)$ of some $j \in J \subseteq S'$ for the basis S. Interpreting the transversal S as a matching in the graph constructed as described above, we see that $C(j, S) = \{j\} \cup A_j \subseteq S'$, as these nodes are the ones which prevent us from finding a matching in the graph G' (i.e., a transversal of $\mathcal{A}'$) when j is added to S. Now, consider WMAP for such an instance. We want to find a cost-minimal subset $X \subseteq J = S' \backslash S$ such that the set $\bigcup_{j \in X} A_j$ covers S, i.e., for all $s \in S$ we have some $j \in X$ such that $s \in C(j, S)$ or equivalently $s \in A_j$. However, this is a general set-covering instance, given that we can start from arbitrary families of subsets of S. As stated before, there is no approximation algorithm with a better approximation factor than $O(\log n)$, unless $\mathsf{P} = \mathsf{NP}$ [41]. □

Therefore, in Conjecture 20, we do not consider the class of representable matroids, but restrict ourselves to the class of regular matroids, which we introduce in the next section.

3.2.3 Totally Unimodular Matrices and Regular Matroids

As seen above, representable matroids are too wide a class for our goal of finding a constant-factor approximation algorithm for WMAP. To see why in Conjecture 20 we focus on regular matroids, let us first consider totally unimodular matrices. A matrix A is *totally unimodular* (TU) if every subdeterminant of A is equal to either -1, 0 or 1, i.e., if we consider any square submatrix B of A, then we have $\det(B) \in \{-1, 0, 1\}$. TU matrices are important for objects in the area of Mathematical Optimization since optimizing over the related polytopes (in the right circumstances) yields integral solutions [42]:

Theorem 23. *Let A be a TU matrix and b be an integral vector of suitable dimension. Then, the polyhedron $P := \{x \in \mathbb{R} \mid Ax \leq b\}$ is integral.*

Tutte [54, 55] has shown that a matroid is regular if and only if it can be represented by the columns of a totally unimodular matrix.

Theorem 24 ([54, 55])**.** *A matroid M is regular if and only if it is $\mathbb{R}$-representable by $\{-1, 0, 1\}$-vectors which together form a TU matrix.*

The two theorems above will be useful later when we express WTAP and WMAP as an IP and use its linear relaxation for the purpose of finding an approximate solution. We will see that the constraint matrix of the linear relaxation forms a TU matrix and that the right-hand side is integral. Thus, we can solve the linear relaxation in polynomial time and still achieve an integral solution.

To do so, let us introduce a special case of TU matrices, namely *network matrices*. Let $D = (V, A)$ be a directed graph and let $T \subseteq A$ be a directed tree on the same

graph. Let $(u, w) = a \in A$ and $t \in T$ and consider the $T \times A$-matrix M defined via

$$M_{t,a} := \begin{cases} 1, & \text{the unique } u\text{-}v \text{ path in } T \text{ passes through } t \text{ in the direction of } t \\ -1, & \text{the unique } u\text{-}v \text{ path in } T \text{ passes through } t \text{ against the direction of } t \\ 0, & \text{else} \end{cases}$$

Such a matrix M is called a network matrix (represented by T and D). See chapter 19.3 in [42] for a proof that network matrices are indeed TU.

In the previous Section 3.2.2, we have introduced regular matroids as matroids which are $\mathcal{F}$-representable over every field $\mathcal{F}$. Note that graphic matroids are regular matroids [42]. Indeed, a matroid is graphic if and only if it can be represented by a network matrix [40]. For a graphic matroid $M(G)$ with a distinguished tree T, pick an arbitrary note $r \in V$ as the root and consider directing all edges in T away from the root. Also, assume all edges in $L = E\backslash T$ are directed arbitrarily. Let A be the network matrix represented by T and (V, L) and let A_l denote the column of A corresponding to $l \in L$. Note the $|A_l|$ is the characteristic vector of $P(l) \subseteq T$. Thus, in particular, the graphic matroid $M(G)$ can be represented by a TU matrix of the form $\begin{bmatrix} I & A \end{bmatrix}$. This can be achieved by taking a network matrix-representation of $M(G)$ and performing Gaussian elimination steps on some related network matrix representing the matroid until the columns related to T form the identity matrix.

Given our discussions on the graphic matroid, we consider a related object, namely the dual matroid to graphic matroids. Let $G = (V, E)$ be a graph and let $\mathcal{I}^\star(G) := \{U \subseteq E \mid E\backslash U \text{ contains a spanning tree of } G\}$. Thus, $M^\star(G) := (E, \mathcal{I}^\star(G))$ is again a matroid, because it is the dual matroid of the graphic matroid $M(G)$. Such matroids are called *cographic matroids*. Note that $B \subseteq E$ is a basis of $M^\star(G)$, if $E\backslash B$ is a spanning tree of G. Additionally, cographic matroids are also regular matroids [42]. Indeed, a matroid is cographic if and only if it can be represented by the transpose of a network matrix [40]. Fix a basis B of $M^\star(G)$ and let $T := E\backslash B$. Consider A as defined above for the case of a graphic matroid (we replace L by B in the definition of course). In particular, a cographic matroid $M^\star(G)$ can be represented by a TU matrix of the form $\begin{bmatrix} I & A^\top \end{bmatrix}$.

In the next Section 3.3, we want to study graphic and cographic matroids in terms of WMAP in more detail as a first step to tackle Conjecture 20.

3.3 Approximating the Graphic and Cographic Case

In this section, we consider WMAP for the cases of graphic and cographic matroids. We show how WMAP can be understood in these two settings and give 2-approximation algorithms for both cases.

3.3.1 The Graphic Matroid Case

For graphic matroids, we have already seen in Section 3.2.1 that WMAP is the same problem as WTAP. The currently best known approximation factor for WTAP is $(1.5 + \varepsilon)$ by Traub and Zenklusen [52]. In this section, we present a more basic 2-approximation algorithm, which is folklore in the field. While not the best known, the 2-approximation will give us some insights on how WMAP behaves for the case of a graphic matroid.

Consider a graph $G = (V, E)$ and a spanning tree $T \subseteq E$. Let $L := E \backslash T$ be the links and consider a cost function $c : L \to \mathbb{R}_{\geq 0}$. Let $M := M(G)$ be the graphic matroid for this instance. The goal is to find a cost-minimal set $S \subseteq L$ such that the fundamental circuits $C(s, T)$ cover T. We have seen in Section 3.2 that this is equivalent to saying that $(V, T \cup S)$ is 2-edge-connected for a cost-minimal set S. Note that for each pair of nodes $u, v \in V$ there is a unique path from u to v in T, which we denote by $P(u, v) \subseteq T$. For $\{u, v\} = l \in L$, we denote $P(u, v)$ alternatively as $P(l)$. Thus, for the fundamental cycle of some $l \in L$ and T we have $C(l, T) = P(l) \cup \{l\}$. We can formulate WTAP on this instance as an IP with binary variables x_l for all $l \in L$. We interpret $x_l = 1$ as $l \in S$ and thus a solution denotes an incidence vector of S.

$$\begin{array}{lllll} \min & \sum_{l \in L} c_l x_l & & & \\ \text{s.t.} & \sum_{l : t \in P(l)} x_l & \geq & 1 & \forall t \in T \\ & x_l & \in & \{0, 1\} & \forall l \in L \end{array} \tag{3.1}$$

By the linear relaxation of the IP 3.1, we denote the LP with the same variables, constraints, and objective function as above but replacing $x_l \in \{0, 1\}$ with $0 \leq x_l \leq 1$ for all $l \in L$.

From G, we choose an arbitrary node $r \in V$ as the root. Given the spanning tree T, for each pair of notes $u, v \in V$ we can define a *least common ancestor*, that is the node that is the predecessor of both u and v which is furthest removed from the root r in terms of the length of the unique path from r. For an edge $\{u, v\} = l \in L$, we denote the least common ancestor as $\operatorname{lca}(l)$. Note that it can happen that $\operatorname{lca}(l) \in \{u, v\}$. Such links are called *uplinks*. Consider a set of links L' obtained from L as follows: For $l \in L$ with $\operatorname{lca}(l) \in \{u, v\}$, we leave l as is. For $l \in L$ with $\operatorname{lca}(l) \notin \{u, v\}$, we bisect $\{u, v\} = l$ at the least common ancestor $\operatorname{lca}(l)$ and thus obtain two edges. More formally,

$$\begin{array}{rcl} L' & := & \{l \in L \mid l = \{u, v\}, \operatorname{lca}(l) \in \{u, v\}\} \\ & \cup & \{\{u, \operatorname{lca}(l)\}, \{\operatorname{lca}(l), v\} \mid l = \{u, v\}, \operatorname{lca}(l) \notin \{u, v\}\}. \end{array}$$

Thus, in L' each link is an uplink. For the new link-set L' we define a new cost-function c' via $c'(l) = c(l)$ if l is an uplink and via $c'(\{u, \operatorname{lca}(l)\}) = c'(\{\operatorname{lca}(l), v\}) = c(l)$ otherwise. For an instance $G = (V, T \cup L')$ and c' of WTAP, we can consider the linear relaxation of the IP 3.1. Consider the constraint matrix A of this LP for the first type of inequalities $\sum_{l : t \in P(l)} x_l \geq 1$ for all $t \in T$. Consider a column of A

for a variable x_l, where $\{u, v\} = l \in L'$. We have seen that l is an uplink. W.l.o.g., assume that $\text{lca}(l) = u$. We have that $A_{t,l} = 1$ for all $t \in P(l)$ and $A_{t,l} = 0$ for all $t \notin P(l)$. Thus, if we interpret T as a directed tree, where each tree-edge $t \in T$ is directed towards the root r and interpret every link in $l \in L'$ as being directed towards $\text{lca}(l)$, we see that A is a network matrix represented by (a directed version of) T and (a directed version of) G. The constraint matrices for the other bounding inequalities are identity matrices. Since it is known that adjoining an identity matrix to a TU matrix results in a TU matrix [42], the entire constraint matrix of the linear relaxation of 3.1 for the instance $G = (V, T \cup L')$ and c' is TU. Thus, since the right-hand side is integral, by Theorem 23, optimizing the LP yields an integral basic optimal solution.

Given this, consider the following Algorithm 5 for WTAP:

Algorithm 5: 2-Approximation Algorithm for WTAP

Input : Graph $G = (V, E)$, spanning tree T, cost function $c : L \to \mathbb{R}_{\geq 0}$ (with $L := E \backslash T$).

Output : A feasible solution x to WTAP with cost at most twice the optimum.

1 Chose $r \in V$ arbitrarily. Compute L' and c' from L and c as described in Section 3.3.1.
2 Solve the LP-relaxation of 3.1 for T, L' and c'. Let $x^\star$ be a basic optimal solution.
3 For each $\{u, v\} = l \in L$ with l_1, l_2 the corresponding bisected edges in L' (if l is an uplink, $l_1 = l_2 = l$), define $x_l := 1$ if $x^\star_{l_1} = 1$ or $x^\star_{l_2} = 1$. Else, $x_l = 0$.
4 Return x.

Lemma 25. *Algorithm 5 is a 2-approximation algorithm for WTAP.*

Proof. Let x be the solution returned by the algorithm on the input $G = (V, E)$, T, $L = E \backslash T$ and c. Let $x^\star$ be the basic optimal solution to the LP for the input T, L' and c' computed by the algorithm in line 2.

Solving the LP can be done in polynomial time, given that we only have polynomially many constraints. All other steps can clearly be completed in polynomial time as well. Thus, the algorithm runs in polynomial time.

As noted above, by Theorem 23 $x^\star$ is integral since the constraint matrix for the LP on the input T, L' and c' is TU and the right-hand side of the LP is integral. Note that in $x^\star$, each tree-edge is covered by some link in L'. Taking a link $l \in L$ and comparing it to its corresponding links $l_1, l_2 \in L'$, we see that the tree-edges covered by l are the union of the tree-edges covered by l_1 and l_2. Since we include l in x whenever l_1 or l_2 are included in $x^\star$, each tree-edge must be covered by x as well since we switch to longer (in terms of the tree-edges covered) links only. Thus, x is a feasible solution.

Lastly, let y be an optimal solution to WTAP on the input T, L and c. Let $y^\star$ be an optimal solution to the linear relaxation of 3.1 on the input T, L and c and let $\bar{y}$ be a solution to the linear relaxation of 3.1 on the input T, L' and c' obtained from $y^\star$, where we set $\bar{y}_{l_1} = \bar{y}_{l_2} = y^\star_l$, with being l_1, l_2 the edges in L' that correspond to $l \in L$. Note that $\bar{y}$ is feasible for the linear relaxation of 3.1 on the input T, L' and c' since $y^\star$ is feasible for the linear relaxation of 3.1 on the input T, L and c. Additionally, $c'(\bar{y}) \leq 2c(y^\star)$ by the definition of $\bar{y}$ since we at most bisect each edge in L and the new bisected edges both have the same cost as the original edge. We have that $c(x) \leq c'(x^\star) \leq c'(\bar{y}) \leq 2c(y^\star) \leq 2c(y)$. The first inequality holds since x contains a link $l \in L$ if $x^\star$ contains at least one of the bisected corresponding links $l_1, l_2 \in L'$, which both have the same cost under c' as l has under c. The second inequality holds since $x^\star$ is an optimal solution to the relevant LP while $\bar{y}$ is a feasible solution to the same LP. The third inequality holds as stated above. The last inequality holds trivially since the minimal solution to the LP relaxation is always smaller than the minimal solution of the IP since we optimize over a smaller space for the IP. □

Thus, we have given a 2-approximation algorithm for WTAP, i.e., for WMAP in the case of a graphic matroid.

3.3.2 The Cographic Matroid Case

Let us recall the definition of the cographic matroid. Let $G = (V, E)$ be a graph. Let $\mathcal{I}^\star(G) := \{U \subseteq E \mid E \backslash U \text{ contains a spanning tree of } G\}$. Then, the matroid $M^\star(G) := (E, \mathcal{I}^\star(G))$ is called the *cographic matroid*. Note that $M^\star(G)$ is the dual matroid of $M(G)$. When the context is clear, we will refer to the cographic matroid only as M in this section.

Let $G = (V, E)$ be a graph, $T \subseteq E$ be a spanning tree and $B := E \backslash T$ be called the *basis-edges*. Let $c : T \to \mathbb{R}_{\geq 0}$ be a cost-function on the tree-edges. Denote by $M = (E, \mathcal{I})$ the cographic matroid for the graph G. Note that B is a basis of M. Consider for some $\{u, v\} = t \in T$ the edge set $T \backslash \{t\}$. The graph $G' := (V, T \backslash \{t\})$ isn't connected and can be understood as two disjoint trees, one rooted in u, the other rooted in v. Let us refer to these trees as T_u and T_v, where $T_u, T_v \subseteq E$. To make G' connected we need to add an edge that connects the two trees T_u and T_v. Let $S_t = S_{\{u,v\}} := \{\{c, d\} = b \in B \mid c \in G[T_u], d \in G[T_v]\}$. Adding any such edge (or t) to the edge set of G' would make the graph connected, i.e., the updated edge set would be a spanning tree of G. Alternatively, already $(V, E \backslash (S_t \cup \{t\}))$ is not connected, i.e., $(V, E \backslash (S_t \cup \{t\}))$ cannot contain a spanning G since there is no way to form a path from u to v. Thus, for the fundamental circuit of t for the basis B we have $C(t, B) = S_t \cup \{t\}$. Therefore, in terms of WMAP, t covers all basis-edges in S_t.

Now, let $\{u, v\} = b \in B$ and denote by $P(b) = P(u, v) \subseteq T$ the unique path in T from u to v. Consider which tree-edges could be used to cover b in terms of WMAP. This is exactly the set $\{t \in T \mid b \in S_t\} = P(u, v)$. Thus, the goal of WMAP in the

case of a cographic matroid can be reformulated as finding a cost-minimal subset $S \subseteq T$ such that for each $\{u, v\} = b \in B$ there is some $s \in S$ with $s \in P(b)$, i.e., there is some $s \in S$ which is part of the unique path in T connecting the endpoints u, v of b.

As mentioned in Section 3.2.3, note that the cographic matroid can be represented by a TU matrix of the form $\begin{bmatrix} I & A^\top \end{bmatrix}$, where A is the network matrix represented by T and (V, B), when interpreting all edges in T as directed away from an arbitrary root $r \in V$ and when all basis-edges are directed arbitrarily.

Next, we will see that WMAP for the case of a cographic matroid is NP-hard. We show this by giving a reduction from the vertex cover problem. In the vertex cover problem, we are given a graph $G = (V, E)$. The goal is to find a minimal subset $S \subseteq V$ such that for each edge $e = \{u, v\}$, S contains at least one of the endpoints u or v, i.e., for all $\{u, v\} = e \in E$ we must have that $u \in S$ or $v \in S$. It is known that the vertex cover problem is NP-hard [25]. Furthermore, the vertex cover problem is APX-hard. More specifically, there is no approximation algorithm with an approximation factor better than $\sqrt{2} - \varepsilon$ for all $\varepsilon > 0$ unless P = NP [28, 12]. Assuming the stronger Unique Games Conjecture holds, it can be shown that no approximation algorithm can have an approximation factor better than 2 [29].

Let $G = (V, E)$ be a graph, an instance of the vertex cover problem. We introduce a new node s a and new set of edges $T := \{\{s, v\} \mid v \in V\}$. We obtain the graph $G' = (V', E')$ with $V' := V \cup \{s\}$ and $E' := E \cup T$. Note that T is a spanning tree of G'. Consider WMAP for the case of the cographic matroid based on the graph G', the spanning tree T, the basis $E = E' \backslash T$ and unit-costs, i.e., $c_t = 1$ for all $t \in T$. For $\{u, v\} = e \in E$ we have $P(e) = P(u, v) = \{\{u, s\}, \{s, v\}\}$. Thus, to cover the fundamental circuit related to e, we must pick one of the tree-edges $\{u, s\}$ or $\{v, s\}$. Thus, if S' is a (cost-)minimal solution to WMAP of the instance described, we obtain a solution S for the vertex cover problem on G by setting $u \in S$ if and only if $\{u, s\} \in S'$. Additionally, if S is a minimal solution to the vertex cover problem on G, we can define $S' := \{\{u, s\} \mid u \in S\}$ to obtain a minimal solution to WMAP on the instance described above.

Let the T, B and c be an instance of WMAP for the case of a cographic matroid as above. We can formulate the problem as an IP. Consider the following binary IP, where $x_t = 1$ denotes that $t \in S$ and thus a solution can be understood as an incidence vector of S.

$$\begin{array}{lllll} \min & \sum_{t \in T} c_t x_t & & & \\ \text{s.t.} & \sum_{t \in P(b)} x_t & \geq & 1 & \forall b \in B \\ & x_t & \in & \{0, 1\} & \forall t \in T \end{array} \tag{3.2}$$

By the linear relaxation of the IP 3.2, we refer to the LP with the same variables, constraints, and objective function as above but replacing $x_t \in \{0, 1\}$ with $0 \leq x_t \leq 1$ for all $t \in T$.

Again, we can choose an arbitrary node $r \in V$ as the root, interpret T as rooted and define the least common ancestor $\operatorname{lca}(u, v)$ for each pairs of nodes $u, v \in V$ and for $\{u, v\} = b \in B$ define $\operatorname{lca}(b) := \operatorname{lca}(u, v)$ as in the previous section. We now consider an approximation algorithm for WMAP in the case of a cographic matroid. For the case of WTAP, in Algorithm 5, we have bisected the links to have uplinks only and thus obtain a constraint matrix which is TU for the LP we solve. Note that for the case of a cographic matroid we cannot just bisect all basis-edges, as by doing so we obtain more and "shorter" edges in the new instance and therefore we might need to include significantly more (costly) tree-edges to cover all fundamental circuits of the bisected basis-edges. Intuitively, in Algorithm 6, we still bisect the basis-edges but only include some of the bisected edges to be able to handle the increase in cost. Therefore, we can obtain an LP with a TU constraint matrix for a slightly altered instance, which we are able to relate back to our original instance. We choose which bisected basis-edges to include based on a basic optimal solution to the linear relaxation of 3.2 for the instance we consider. To state this more formally, consider Algorithm 6.

Algorithm 6: 2-Approximation Algorithm for WMAP for the case of a cographic matroid

Input : Graph $G = (V, E)$, basis B, cost function $c : T \to \mathbb{R}_{\geq 0}$ (with $T := E \backslash B$).

Output : A feasible solution to WMAP for the given input with cost at most twice the optimum.

1 Chose $r \in V$ arbitrarily. This is needed to define the least common ancestor of pairs of notes.
2 Solve the LP-relaxation of 3.2 for T, B and c. Let $x^\star$ be a basic optimal solution.
3 Define $B' := \{b \mid \{u, v\} = b \in B \text{ with } \operatorname{lca}(b) \in \{u, v\}\} \cup \left\{\{u, \operatorname{lca}(b)\} \mid \{u, v\} = b \in B, \operatorname{lca}(b) \notin \{u, v\}, \sum_{t \in P(u, \operatorname{lca}(b))} x^\star_t \geq \frac{1}{2}\right\}$.
4 Solve the LP-relaxation of 3.2 for T, B', c. Let x' be a basic optimal solution.
5 Return x'.

Theorem 26. *Algorithm 6 is a 2-approximation algorithm for WMAP for the case of a cographic matroid.*

Proof. Let x' be the solution returned by the algorithm on the input $G = (V, E)$, T, $B = E \backslash T$ and c. Let $x^\star$ be the basic optimal solution to the first LP solved by the algorithm in line 2.

Solving both LPs can be done in polynomial time, given that we only have polynomially many constraints in both cases because the cardinality of B' is bounded

by twice the cardinality of B. All other steps can clearly be completed in polynomial time as well. Thus, the algorithm runs in polynomial time in terms of the input size.

Consider $x^\star$ and some arbitrary $\{u, v\} = b \in B$. Say $\operatorname{lca}(b) = w$. We first consider the case that $u, v \neq w$, i.e., that b is not an uplink. Since $x^\star$ is a feasible solution to the LP, we must have $\sum_{t \in P(u,v)} x^\star_t = \sum_{t \in P(u,w)} x^\star_t + \sum_{t \in P(w,v)} x^\star_t \geq 1$. Therefore, $\sum_{t \in P(u,w)} x^\star_t \geq \frac{1}{2}$ or $\sum_{t \in P(w,v)} x^\star_t \geq \frac{1}{2}$ must hold. Thus, in this case note that for a basis-edge in B we have at most two corresponding edges in B'. Next, if b is an uplink, assume w.l.o.g. that $w = u$. By definition we have that $b \in B'$ and therefore there is one edge in B' that corresponds to the original uplink. Therefore, we have that all edges in B' are uplinks as defined in the last section. Consider the constraint matrix A for the first type of inequalities $\left(\sum_{t \in P(b)} x_t \geq 1 \text{ for all } b \in B'\right)$ of the second LP solved by the algorithm in line 4 for the input T, B' and c. Consider a row of A for an edge $\{u, v\} = b' \in B'$. As noted above, b' is an uplink. W.l.o.g., we assume $\operatorname{lca}(b') = u$. We show that A is the transpose of a network matrix. Similar as in the proof of lemma 25, we have $A_{b',t} = 1$ for all $t \in P(b')$ and $A_{b',t} = 0$ for all $t \notin P(b')$. Therefore, we can interpret T as a directed tree, where each tree-edge t is directed towards the root r and can interpret every basis-edge $b' \in B'$ as being directed towards $\operatorname{lca}(b')$. By doing so, we see that A is indeed the transpose of a network matrix represented by (a directed version of) T and (a directed version of) G. Thus, as in the proof of lemma 25, the entire constraint matrix is TU [42], for the input T, B' and c. Also, the right-hand side of the LP is integral. Thus, by Theorem 23, x' is an integral (binary) vector. Therefore, x' is a feasible solution to the IP with inputs T, B' and c, i.e., x' hits all edges in B' as described earlier in this section. Note that each edge in B is at least as long as the corresponding edges in B' (in terms of the length of the unique path in T). Thus, when replacing any edge $b' \in B'$ by the corresponding longer edge $b \in B$, x' still hits b since $P(b') \subseteq P(b)$. Therefore, x' is also a feasible (integral) solution to WMAP for the cographic matroid on the instance T, B and c.

Let y be defined via $y_t := \min\{1, 2x^\star_t\}$ for all $t \in T$ and let z be an optimal solution to the IP 3.2 for the input T, B and c. Note that y is a feasible solution to the linear relaxation of 3.2 with inputs T, B' and c. To see this, first consider some $\{u, v\} = b \in B$ with $\operatorname{lca}(b) \notin \{u, v\}$. Say that $\operatorname{lca}(b) = w$. We have $\sum_{t \in P(u,w)} x^\star_t \geq \frac{1}{2}$ or $\sum_{t \in P(v,w)} x^\star_t \geq \frac{1}{2}$. W.l.o.g. let us assume the former. If $y_t = 1$ for some $t \in P(u, w)$, we have $\sum_{t \in P(u,w)} y_t \geq 1$, as desired. Else, we have $\sum_{t \in P(u,w)} y_t = \sum_{t \in P(u,w)} 2x^\star_t \geq 1$. Else, if $\operatorname{lca}(b) = u$, we have $\sum_{t \in P(u,v)} x^\star_t \geq 1$. Again, if for some $t \in P(u, v)$ we have $y_t = 1$ we are done. Else, we obtain $\sum_{t \in P(u,v)} y_t = \sum_{t \in P(u,v)} 2x^\star_t \geq 1$ and the result follows as well. Also, y clearly satisfies the box constraints, i.e., $0 \leq y_t \leq 1$ for all $t \in T$. We have $c(x') \leq c(y) \leq 2c(x^\star) \leq 2c(z)$. The first inequality holds since y is a feasible solution to the LP with input T, B' and c, while x' is an optimal solution for the same LP. The second inequality holds since $y_t \leq 2x^\star_t$ for all $t \in T$ by the definition of y. The third inequality holds since $x^\star$ is optimal for the linear relaxation while z is an optimal solution for the IP. □

3.4 IP-Formulations and Iterative Rounding

In this section, we consider an IP-formulation for the general WMAP problem, give some observations on this formulation, and discuss a first potential approach on how to design an approximation algorithm for WMAP for the case of regular matroids.

3.4.1 An IP-Formulation of WMAP and some Observations

Consider a TU-matrix $A' \in \{-1, 0-1\}^{m \times n}$ with $m \leq n$ and full row-rank. It is known [42] that applying Gaussian elimination (via pivoting operations [42]) to a TU matrix preserves total unimodularity. Thus, we can assume that $A' = \begin{bmatrix} I & A \end{bmatrix}$, where I is the $m \times m$ identity matrix and A is some other TU matrix. By Theorem 24, A' can be understood as a $\mathbb{R}$-representation f of some regular matroid M with n elements and where the size of any basis is m. W.l.o.g., we denote the elements of M as $[n]$, where $1 \leq k \leq n$ is represented by A'_k, the k^{th} column of A'. Note that $B = [m]$ is a basis of M, as the elements $\{1, 2, ..., m\}$ are linearly independent since their images together form the $m \times m$-identity matrix. We define $N := E \backslash B$. Given this, we have $A' = \begin{bmatrix} I & A \end{bmatrix}$, where I is the image of the basis B and A is the image of the elements in N. Consider some element $k \in N$ and let $a := A_k = A'_{m+k}$ be the k^{th} column of A representing the element k. Note that $a - \sum_{i=1}^{m} a_i e_i = a - \sum_{i=1}^{m} a_i A'_i = \mathbb{0}$, where a_i is the i^{th} entry of a. Thus, $C(k, B) = \operatorname{supp}\{a\} \cup \{k\}$, where $\operatorname{supp}\{a\}$ denotes the support of a, which are all entries with a value different from 0.

Let us now consider an IP-formulation for WMAP. The goal is finding a cost-minimal $S \subseteq N$ such that the set $\{C(s, B) \mid s \in S\}$ covers B. Consider the following binary IP[3], where $x_k = 1$ for $k \in N$ is interpreted as $k \in S$ and thus a solution can be understood as an incidence vector of S.

$$\begin{array}{lllll} \min & \sum_{k \in N} c_k x_k & & & \\ \text{s.t.} & \sum_{k : b \in C(k,B)} x_k & \geq & 1 & \forall b \in B \\ & x_k & \in & \{0,1\} & \forall k \in N \end{array} \tag{3.3}$$

By the linear relaxation of 3.3 we refer to the LP with the same variables, constraints, and objective function as above but replacing $x_k \in \{0, 1\}$ with $0 \leq x_k \leq 1$ for all $k \in N$.

Consider a regular matroid M which is represented by a TU matrix $A' = \begin{bmatrix} I & A \end{bmatrix}$ as above. Let Q be the constraint matrix of the linear relaxation of 3.3 for the input M for the first type of inequalities $\sum_{k : b \in C(k,B)} x_k \geq 1$ for all $b \in B$. Consider Q_k, the column of Q which is related to the variable x_k. We have $Q_{b,k} = 1$ for all $b \in B$ with $b \in C(k, B)$ and $Q_{b,k} = 0$ for all $b \in B$ with $b \notin C(k, B)$. Thus, $Q_{b,k} = 1$ if and only if $b \in \operatorname{supp}\{A_k\}$. It follows that we can understand Q as the absolute value of A, i.e., Q is obtained by taking the absolute value of a TU matrix which

[3] Note that we can compute all inequalities of 3.3 in polynomial time from the input $M = (E, \mathcal{I})$ with a distinguished basis B, even if the matrix A is not given since we can find the fundamental circuits $C(k, B)$ for each $k \in N$ in polynomial time [37].

represents the non-basis elements of a regular matroid. Note that from the points above can can conclude that both statements of our main conjecture in this chapter (Conjecture 20 and Conjecture 21) are equivalent.

3.4.2 Introduction to Iterative Rounding

Iterative Rounding is a method to round LP-solutions which was introduced by Jain [23]. It is the first approach we tried to round solutions of the linear relaxation of 3.3 to design a suitable approximation algorithm. We first give a short overview of iterative rounding as introduced by Jain [23]. Then, we apply this method to the graphic and cographic case in the next Section 3.4.3. The algorithm and the proofs shown in this section follow Jain [23] but are slightly adapted to the problem at hand.

The idea of iterative rounding to obtain a 2-approximation algorithm, as introduced by Jain [23], is based on three premises[4]:

- We have an LP to which we can find a basic optimal solution in polynomial time on all inputs. This is e.g., because the LP has polynomial size in terms of the input size or because we can efficiently solve the separation problem.
- When fixing some variables to integer values (for binary IPs to 0 or 1) and considering the reduced problem[5], it has the same form as the original LP, i.e., it is the same LP for some different (reduced, or – for some versions of iterative rounding – generalized) instance of the same problem.
- In each basic optimal solution to the relevant LP, there is at least one variable that takes value at least $\frac{1}{2}$.
  Note that this last premise is not required for many generalizations and adaptations of iterative rounding, where we e.g., aim for different approximation factors than 2. In those cases, we only want there to be some variable with a value above some fixed threshold. However, the premise as stated here is required for iterative rounding as first introduced by Jain [23] and to obtain a 2-approximation algorithm, which is why we state it in the form seen above.

Based on the above, we want to give a short and informal overview of the procedure, following Jain [23]. If the three premises above are satisfied, we start by solving the LP for the original input. We set all variables to 1 which take a value of at least $\frac{1}{2}$ in the basic optimal solution obtained from solving the LP. By doing so, we can adjust the right-hand sides as some variables are now fixed and obtain a smaller instance of the same LP and can thus iterate the process. Given that there is always at least one variable in each iteration with value of at least $\frac{1}{2}$, the algorithm will terminate in

[4]Note that there are various generalizations and adaptions of the procedure [15, 34, 36].

[5]The reduced problem is usually attained by introducing new constraints fixing the variables mentioned before. Alternatively, one can fix every occurrence of the variables to their prescribed value, adjust the right-hand side accordingly, remove the variables from the IP and drop any already satisfied constraints.

polynomial time since we can solve each individual LP in polynomial time. Since in every iteration we only double the contribution to the cost by increasing the values of some variable from at least $\frac{1}{2}$ to 1, we get a 2-approximation algorithm.

To be more formal, we consider some IP 3.4, for which we assume that some assumption, stated below, hold:

$$\begin{array}{lllll} \min & \sum_{e\in E} c_e x_e & & & \\ \text{s.t.} & \sum_{e\in E} A_{d,e} x_e & \geq & 1 & \forall d \in D \\ & x_e & \in & \{0,1\} & \forall e \in E \end{array} \tag{3.4}$$

For notational convenience, we assume $A \in \{0,1\}^{D\times E}$. We consider the linear relaxation to 3.4, where we replace the integrality constraint by $0 \leq x_e \leq 1$ for all $e \in E$. Assume for now that for every input D, E, A and c in every basic optimal solution x of the linear relaxation, there exists some $e \in E$ such that $x_e \geq \frac{1}{2}$. Let $x^\star$ be such a basic optimal solution. Clearly, the linear relaxation has polynomial size in terms of the input size and can thus be solved in polynomial time. Consider some $S \subseteq E$. If we set $x_s = 1$ for all $s \in S$, we obtain a residual problem which can be formulated as an LP of the same form (with appropriately restricted versions of A and c) on the input $E' := E\backslash S$ and $D' := D\backslash \left\{d \in D \mid \sum_{s\in S} A_{d,s} \cdot 1 \geq 1\right\}$. That is, we fix the variables in S to 1 and drop the constraints which are already satisfied by the fixed variables. Going forward, we will always use $S := \left\{s \in E \mid x^\star_s \geq \frac{1}{2}\right\}$. Therefore, all three premises stated above are satisfied and we are thus in the setting where iterative rounding can be applied.

Given this, iterative rounding can be formulated as Algorithm 7.

Algorithm 7: Iterative Rounding [23]

Input : $D, E, A \in \{0,1\}^{D\times E}, c$.

Output : A feasible integral solution to the IP 3.4 with cost at most twice the optimum.

```
Set S ← ∅.
while D ≠ ∅ do
    Solve the linear relaxation of 3.4 with input E and D. Let x* be a
      basic optimal solution.
    Let S* := {s ∈ E | x*_s ≥ 1/2}.
    Set S ← S ∪ S*, E ← E\S* and
      D ← D\{d ∈ D | Σ_{s∈S*} A_{d,s} · 1 ≥ 1}.
    Set A and c to appropriately restricted versions of themselves.
end
Return S.
```

To analyze Algorithm 7, consider the following Lemma 27.

Lemma 27 ([23])**.** *Let $x^\star$ be the optimal solution to the linear relaxation of 3.4 on the input E, D. Let $S^\star := \left\{s \in E \mid x^\star_s \geq \frac{1}{2}\right\}$, $D' = D\backslash \left\{d \in D \mid \sum_{s\in S^\star} A_{d,s} \cdot 1 \geq 1\right\}$ and let $E' = E\backslash S^\star$. Let $x^\star_{res}$ be the optimal solution to the linear relaxation of 3.4*

on the input E', D'. Lastly, let S_{res} be a feasible integer solution to the IP 3.4 on the input E', D' with cost at most twice the cost of $x^\star_{res}$.

Then, $S^\star \cup S_{res}$ is an integral solution to the IP 3.4 on the input E, D with $c(S^\star \cup S_{res}) \leq 2c(x^\star)$.

Proof. Note that S_{res} satisfies all inequalities in D' by assumption while $S^\star$ satisfies all inequalities in $D \backslash D'$ by definition of $S^\star$ and D'. Thus, $S^\star \cup S_{res}$ is a feasible integral solution to the IP 3.4 with input E, D.

Note that $\sum_{e \in E'} A_{d,e} x^\star_e \geq 1$ for all $d \in D'$ as well. Thus $x^\star$ restricted to the variables in E' is also a feasible solution to the linear relaxation of 3.4 with input E', D'. Therefore, we have

$$c(x^\star_{res}) \leq c(x^\star) - \sum_{e \in S^\star} c_e x^\star_e.$$

With this, we get

$$\begin{array}{rcll} 2c(x^\star) & \geq & 2c(x^\star_{res}) + 2\sum_{e \in S^\star} c_e x^\star_e & \text{by the inequality above} \\ & \geq & 2c(x^\star_{res}) + \sum_{e \in S^\star} c_e & \text{since } 2x^\star_e \geq 1 \text{ for all } e \in S^\star \text{ by definition} \\ & \geq & c(S_{res}) + c(S^\star) & \text{since } c(S_{res}) \leq 2c(x^\star_{res}) \text{ by assumption} \\ & = & c(S_{res} \cup S^\star) & \end{array}$$

□

Thus, we obtain Lemma 28.

Lemma 28 ([23])**.** *Algorithm 7 is a 2-approximation algorithm to the IP 3.4 given the assumptions stated above.*

Proof. Solving each individual LP can be done in polynomial time. All other steps can be completed in polynomial time as well. Given the third premise, in each step there is always at least one variable with value of at least $\frac{1}{2}$. Thus, the main loop is executed at most $|D|$ times. Therefore, the algorithm runs in polynomial time in terms of the input size.

We now use iteration over the at most $|D|$ executions of the main loop of the algorithm, repeatedly applying Lemma 27, to show that in each step we obtain a 2-approximation algorithm for the reduced version of the IP 3.4 considered in that step. Thus overall, it follows that S is a feasible solution to the IP 3.4 and that the cost of S is at most twice the cost of the optimum of the IP. □

Now that we have seen how iterative rounding works, we can try to apply it to WMAP. Note that our IP 3.3 has polynomially many constraints and can thus be solved in polynomial time. Additionally, if we set the variables related to some edges $N' \subseteq N$ to 1, we cover some elements $B' \subseteq B$. The new, smaller instance of the problem thus obtained can be expressed as the same general IP for the input $N \backslash N'$ and $B \backslash B'$. Thus, in the next Section 3.4.3 we consider if we can always find a variable with value at least $\frac{1}{2}$ for the case of WMAP, i.e., if the third premise is satisfied.

3.4.3 Iterative Rounding for the Graphic and Cographic Case

In this section, we show that iterative rounding as described in the previous section works for the case of WTAP to obtain a 2-approximation algorithm. Note that this result is well-known in the field. We here recontextualize the approach and reprove the statement. However, for WMAP in the case of a cographic matroid, we show that the IP 3.2 does not satisfy the third premise for iterative rounding as stated in the previous section.

First, we consider WMAP for the case of a graphic matroid, which as we have seen is equivalent to WTAP. Recall the IP 3.1.

Lemma 29. *Let y be a basic optimal solution to the linear relaxation of the IP 3.1 for some input T, L, c. Then, there exists some $l \in L$ such that $y_l \geq \frac{1}{2}$.*

Proof. Assume for a contradiction that the statement is false. Consider a lexicographical ordering of all possible instances where we primarily sort by $n := |T|$ and, among instances with the same n, secondarily by $m := |L|$. Consider the lexicographically smallest instance of WTAP such that for some vector c there exists a basic optimal solution x where we have $x_l < \frac{1}{2}$ for all $l \in L$. Let T, L, c and y be an instance and a basic optimal solution witnessing the above. Note that since T is a tree, the related graph has $n + 1$ nodes.

We now consider some properties which must hold for the minimal counterexample T, L, c and y. First, we must have $y_l > 0$ for all $l \in L$. Else, we could remove the links whose variables take value 0 and consider the instance $T, \{l \in L \mid y_l > 0\}, c$. This would be a lexicographically smaller instance with the same properties as mentioned above. We therefore have that $m \leq n = |T|$ since we have n constraints and thus at most n variables must take non-zero values in any basic solution. Furthermore, we can assume that $\sum_{l:t \in P(l)} y_l = 1$ for all $t \in T$, as if we had a non-tight such constraint, we could drop it and y would still remain a basic optimal solution. Note that dropping a constraint is equivalent to contracting the related tree. Thus, from having such a non-tight constraint we could construct a lexicographically smaller instance with the same properties as mentioned above, meaning that all such constraints must indeed be tight, to not contradict minimality of T, L, c and y.

Let some arbitrary node $r \in V$ be the root. Recall that our graph has $n + 1$ nodes. We first consider the case where every node either has at least two direct descendants (i.e., descendants that are adjacent to the node) or is a leaf. Note that in such a tree at least half the nodes must be leaves. Consider a tree-edge $\{u, v\} = t \in T$ which is adjacent to a leave u. Clearly, this tree-edge t can only be covered by links $l \in L$ which are adjacent to u. However, since $y_l < \frac{1}{2}$ for all $l \in L$, there must be at least three links adjacent to u to collectively cover the tree-edge t, i.e., to satisfy the constraint $\sum_{l:t \in P(l)} y_l \geq 1$. Since this holds for every leaf u, we must have at least $3\frac{n+1}{2} > n$ links in L. Because we however have $|L| = m \leq n$, no such instance can exist, leading to a contradiction.

Now, consider the case where we have a node $u \in V$ with $u \neq r$ which is adjacent

to exactly two other nodes via tree-edges[6]. I.e., u is incident to exactly two tree-edges $t_1 = \{v, u\}, t_2 = \{u, w\} \in T$ and thus u has exactly one direct predecessor v and exactly one direct descendant w. Let $In(u) := \{\{a, u\} = l \in L \mid t_1 \in P(l), t_2 \notin P(l)\}$, i.e., the set of links that are incident to u which cover t_1 but not t_2. Similarly, we define $Out(u) := \{\{u, b\} = l \in L \mid t_1 \notin P(l), t_2 \in P(l)\}$ and define $Over(u) := \{l \in L \mid t_1 \in P(l), t_2 \in P(l)\}$. Note that $\sum_{l \in In(u)} y_l = \sum_{l \in Out(u)} y_l$ since we have $\sum_{l: t_1 \in P(l)} y_l = 1 = \sum_{l: t_2 \in P(l)} y_l$ and thus it follows that $\sum_{l \in In(u))} y_l + \sum_{l \in Over(u))} y_l = 1 = \sum_{l \in Out(u))} y_l + \sum_{l \in Over(u))} y_l$.

By splitting a link $\{a, b\} = l \in In(u) \dot\cup Out(u)$ we mean replacing l with two new parallel links $l_1 = \{a, b\}, l_2 = \{a, b\}$ and replacing the related variable y_l by y_{l_1}, y_{l_2} such that $y_l = y_{l_1} + y_{l_2}$ and $0 \leq y_{l_1}, y_{l_2} \leq 1$. Note that there are different ways to split a link, depending on how we distribute the value of y_l between y_{l_1} and y_{l_2}. In any case, we define $c_{l_i} := c_l \cdot \frac{y_{l_i}}{y_l}$ for $i = 1, 2$. Thus, $c_l = c_{l_1} + c_{l_2}$, i.e., splitting the link leads to a larger instance with an extended solution with the same cost as the original solution. Now, consider the sets $In'(u)$ and $Out'(u)$ obtained from splitting the links in $In(u)$ and $Out(u)$ such that we have $|In'(u)| = |Out'(u)|$ and such that there exists a bijection $\mu : In'(u) \to Out'(u)$ where for all $l \in In'(u)$ we have $y_l = y_{\mu(l)}$, i.e., we split the links in $In(u)$ and $Out(u)$ in such a way that we can map splitted links whose variable take the same value to each other. Note that we can always achieve such a splitted instance. Now, consider some $\{a, u\} = l \in In'(u)$ and $\{u, b\} = \mu(l) \in Out'(u)$. We can combine the two splitted links into a longer link $l' = \{a, b\}$ and replace the variables $y_l, y_{\mu(l)}$ by $y_{l'} := y_l = y_{\mu(l)}$. We define $c_{l'} := c_l + c_{\mu(l)}$. Thus $c_{l'} y_{l'} = c_l y_l + c_{\mu(l)} y_{\mu(l)}$, i.e., combining the splitted links in such a way leads to a new instance with a solution with the same cost as the cost for the solution for the splitted instance. Consider the instance obtained as described by splitting and combining the links in $In(u)$ and $Out(u)$. Note that after splitting and recombining links as described, in the new instance no link is adjacent to u anymore. Thus, we can contract the edges t_1 and t_2. Therefore, we arrive at a lexicographically smaller instance of the problem, as we have decreased $|T|$ (while $|L|$ might have become much larger). This, however, contradicts our assumption of considering the smallest counterexample. Therefore, every node must have at least two direct descendants or be a leaf which puts us back into the setting considered before and the lemma follows. □

Thus, for WTAP with the IP 3.1, iterative rounding works as described in Section 3.4.2.

Next, we consider WMAP for the case of a cographic matroid with the IP 3.2.

[6] We still assume that the root r has at least two direct descendants. Note that this assumption is w.l.o.g.. We do not consider the case that r is the only node, as here no edges exist. If r has exactly one direct descendant r', we can instead assume that r' is the root. If the new root r' now has at least two direct descendants, we are done. If again r' has only one direct descendant r, the tree consists of a single tree-edge. In this case, in any basic solution we must necessarily choose a single link completely, thus we cannot have $y_l < \frac{1}{2}$ for all $l \in L$. Therefore, the assumption is w.l.o.g..

Consider the instance shown in Figure 3.1 below. The colorful **full** edges are the basis-edges (the different colors are only used to (hopefully) enhance readability and don't have any further meaning). The black dashed edges are the tree-edges. Note that the basis-edges also form a spanning tree which is edge-disjoint from the tree-edges. Fix an arbitrary basis-edge b and consider $P(b)$, as defined in Section 3.3.2. Note that $|P(b)| = 3$. Thus, by setting $x_t = \frac{1}{3}$ for all $t \in T$ we have $\sum_{t \in P(b)} x_t = 3 \cdot \frac{1}{3} = 1$ for each basis-edge b. Therefore, this solution is feasible. Furthermore, we have exactly $|B| = 14$ tight constraints and exactly $|T| = 14$ variables. It follows that the solution is basic. Additionally, it can be checked that for $c = \mathbb{1}$, i.e., the all-ones vector, the solution is optimal. Overall, in Figure 3.1 we have found an instance of WMAP for the cographic matroid with a basic optimal solution where $x_t < \frac{1}{2}$ for all $t \in T$. Therefore, iterative rounding as described in Section 3.4.2 cannot be applied to WMAP in the case of a cographic matroid using the IP 3.2 to obtain a 2-approximation algorithm.

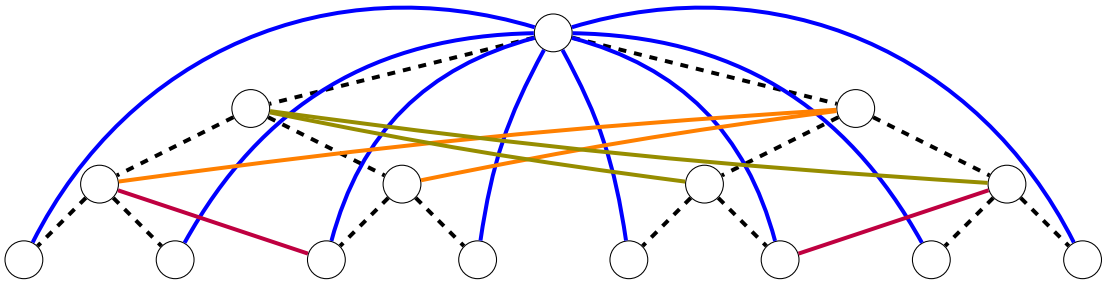


Figure 3.1

Thus, for an iterative rounding-based approach to work, we would have to modify Algorithm 7 or consider a different type of IP, i.e., by using a different formulation or by somehow adjusting some inequalities between iterations. We leave this approach open for now as a jumping off point for further research.

3.5 Seymour Decomposition

In this section, we present an alternative approach on how to potentially design an approximation algorithm for WMAP for the case of regular matroids. First, let us consider the following definitions. Let

$$M_{10} := \begin{bmatrix} 1 & -1 & 0 & 0 & -1 \\ -1 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 1 \end{bmatrix}$$

It can be shown that M_{10} is TU [42]. We refer to the matroid represented by $\begin{bmatrix} I & M_{10} \end{bmatrix}$ as the matroid R_{10}.

Next, we define three matroid operations, which take two matroids as inputs, namely the *one-sum* $\oplus_1$, *two-sum* $\oplus_2$ and *three-sum* $\oplus_3$. These operations will have the property that they preserve matroid regularity, i.e., if both inputs are regular, the resulting matroid will also be regular [46]. Since in our context, we only care about regular matroids, we primarily define and use these operations in terms of the TU matrices representing the regular matroids, i.e., we define and use the operations as operations on TU matrices, following the notation of [42]. Let M, N be regular matroids represented by some TU matrices $\begin{bmatrix} I & A \end{bmatrix}$ and $\begin{bmatrix} I & B \end{bmatrix}$. Thus, we then define $K = M \oplus_i N$ as the matroid that is represented by $\begin{bmatrix} I & C \end{bmatrix}$, where $C := A \oplus_i B$ for $i = 1, 2, 3$.

Let A, B be TU matrices and let a and d be $\{-1, 0, 1\}$-column vectors and b and c be $\{-1, 0, 1\}$-row vectors of suitable dimensions. All matrices shown below on the left-hand side (as inputs) are assumed to be TU, e.g., $\begin{bmatrix} A & a \end{bmatrix}$ as an input for the 2-sum is assumed to be TU. The outputs on the right-hand side can, as mentioned, also shown to be TU. By $\mathbb{0}$ we denote the all zero-matrix of suitable dimensions. We have

$$A \oplus_1 B := \begin{bmatrix} A & \mathbb{0} \\ \mathbb{0} & B \end{bmatrix} \tag{3.5}$$

$$\begin{bmatrix} A & a \end{bmatrix} \oplus_2 \begin{bmatrix} b \\ B \end{bmatrix} := \begin{bmatrix} A & ab \\ \mathbb{0} & B \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} A & a & a \\ c & 0 & 1 \end{bmatrix} \oplus_3 \begin{bmatrix} 1 & 0 & b \\ d & d & B \end{bmatrix} := \begin{bmatrix} A & ab \\ dc & B \end{bmatrix} \tag{3.7}$$

To gain some further insights into these operations, we consider the following, as stated by Seymour [46]. Under certain conditions to be stated later, the one-, two- and three-sums of graphic matroids are graphic matroids themselves. Therefore, the operations can be understood in terms of graphs as well in certain cases. Let $G = (V, E)$ be a connected graph. $Y \subseteq V$ is called a *cut-set* if removing the nodes in Y and all adjacent edges causes the graph to be disconnected. Let Y be a minimal cut-set of G and choose $T_1, T_2 \neq \emptyset$ such that (T_1, Y, T_2) is a partition of V and such that no edge in E joins a node of T_1 to a node of T_2. Let $Z := \{\{y, z\} \mid y \neq z, y, z \in Y\}$, i.e., let Z be a (potentially new) set of edges joining each pair of nodes in Y. If $|Y| = k$ for $k = 1, 2, 3$ and if we can find two subgraphs $G_i = (V_i, E_i)$, for $i = 1, 2$ such that:

- $V_i = Y \cup T_i$
- $E_1 \cup E_2 = E \cup Z$
- $E_1 \cap E_2 = Z$
- $|E_1|, |E_2| < |E|$

then the graphic matroid $M(G)$ is the k-sum of the graphic matroids $M(G_1)$ and $M(G_2)$.

Given these definitions, we can state a seminal result by Seymour [46] which characterizes all regular matroids:

Theorem 30 ([46])**.** *Let N be a regular matroid. There exists a finite family $\mathcal{M} := (M_i)_{i\in I}$ of matroids such that N can be constructed from members of $\mathcal{M}$ be repeatedly applying* 1*-,* 2*- and* 3*-sum operations. Furthermore, for all $i \in I$, M_i is either a graphic matroid, a cographic matroid or the matroid R_{10}.*

Given this result, we can consider the following approach to prove Conjecture 20. We know from previous sections that there is a 2-approximation algorithm for WMAP for the case of a graphic and a cographic matroid. Also, we can solve the case for R_{10} to optimality since it is only a single matroid. If we can show for the one-, two- and three-sum that if there is a 2-approximation algorithm for both inputs then there is also a 2-approximation algorithm for the output, we can use Theorem 30 to conclude that there is a 2-approximation algorithm for WMAP for every regular matroid.

Let $\begin{bmatrix} P \\ I \end{bmatrix}$ be the constraint matrix of the IP 3.3 from Section 3.4.1 on the input of a regular matroid M. Recall that P can be understood as the absolute value of a matrix A such that M is represented by $\begin{bmatrix} I & A \end{bmatrix}$. Given this, the can restate what we have to show as follows: For $i = 1, 2, 3$, let M, N, O be regular matroids which are represented by $\begin{bmatrix} I & A \end{bmatrix}$, $\begin{bmatrix} I & B \end{bmatrix}$ and $\begin{bmatrix} I & C \end{bmatrix}$ such that $O := M \oplus_i N$, i.e., $C = A \oplus_i B$. Set $P := |A|$, $Q := |B|$ and $R := |C|$. Let B_M, B_N and B_O be the bases of M, N, O identified by the choice of the matroid representations $\begin{bmatrix} I & A \end{bmatrix}$, $\begin{bmatrix} I & B \end{bmatrix}$, $\begin{bmatrix} I & C \end{bmatrix}$ and let c_M, c_N and $c_O = (c_M, c_N)^\top$ be cost-vectors of suitable dimensions. Let $x^\star_M, x^\star_N$ and $x^\star_O$ be optimal solutions to WMAP on the instances M, B_M, c_M, N, B_N, c_N and O, B_O, c_O. Assume that in polynomial time we can compute feasible solutions x_M and x_N such that $c_M(x_M) \leq 2c_M(x^\star_M)$ and $c_N(x_N) \leq 2c_N(x^\star_N)$. We now want to show that we can find in polynomial time a feasible solution x_O such that $c_O(x_O) \leq 2c_O(x^\star_O)$.

Let us first consider the case of the one-sum. We show that Conjecture 20 holds when considering only one-sums.

Lemma 31. *Let $\bar{M}$ be a matroid such that there exists a finite family $\bar{\mathcal{M}} := (\bar{M}_i)_{i\in I}$ of matroids where for all $i \in I$, $\bar{M}_i$ is either a graphic matroid, a cographic matroid or the matroid R_{10} and such that $\bar{M}$ can be constructed from members of $\bar{\mathcal{M}}$ be repeatedly applying the* 1*-sum operation. Then, there is a* 2*-approximation algorithm for WMAP in the case of an input N.*

Proof. We have seen in Lemma 25 and Theorem 26 that there is a 2-approximation algorithm for WMAP if the matroid is graphic or cographic. Furthermore, WMAP

with the matroid R_{10} is a single optimization problem, which we can solve to optimality. Thus, to show the statement, it suffices to prove that if we have two regular matroids for which we have a 2-approximation algorithm for WMAP, then there is a 2-approximation algorithm for their one-sum, as we then can apply this statement repeatedly constructing $\bar{M}$ from $\bar{\mathcal{M}}$.

Let M, N and O be regular matroids and let all related notation be as before. By assumption, we can in polynomial time compute suitable approximate solutions x_M and x_N. We now set $x_O := (x_M, x_N)^\top$. From the definition of the one-sum 3.5, we see that x_O is feasible for WMAP on the input O, B_O, c_O, as $\begin{bmatrix} P & \mathbb{0} \end{bmatrix} (x_M, x_N)^\top = Px_M \geq \mathbb{1}$ since x_M is feasible for WMAP on the input M, B_M, c_M. Similarly, $\begin{bmatrix} \mathbb{0} & Q \end{bmatrix} (x_M, x_N)^\top \geq \mathbb{1}$. Note that can assume that $x_O^\star = (x_M^\star, x_N^\star)^\top$ since by using the same argument as above, we can split x_O into two parts x_O^M and x_O^N. Note that for x_O to be feasible for the input O, B_O, c_O, we must have that x_O^M and x_O^N are feasible for the inputs M, B_M, c_M and N, B_N, c_N respectively. Thus, we can replace x_O^M and x_O^N by $x_M^\star$ and $x_N^\star$ without increasing the cost. Additionally, $c_O(x_O) = (c_M, c_N)^\top((x_M, x_N)^\top) = c_m(x_M) + c_N(x_N) \leq 2\left(c_M(x_M^\star) + c_N(x_N^\star)\right) = 2c_O(x_O^\star)$ by our assumptions. Thus, the statement follows. □

For the two-sum and for the three-sum, the problem becomes more involved. Let $x_O := (x_M, x_N)^\top$ as above. Note that x_O is again a feasible solution. However, the argument about splitting x_O into x_O^M and x_O^N does not work any longer, as the less sparse output matrix means that both partial solutions "interact" with each other so x_O^M and x_O^N must not necessarily be feasible solutions for the inputs M, B_M, c_M and N, B_N, c_N anymore. Thus, we cannot assume that $x_O^\star = (x_M^\star, x_N^\star)^\top$. Given the IP 3.3, we can always find in polynomial time an optimal solution to the linear relaxation. The question remains, how to suitably round said solution.

Given its less involved definition, it seems reasonable to next consider the case of the two-sum. Let $M = \begin{bmatrix} A & a \end{bmatrix}$ and $N = \begin{bmatrix} b \\ B \end{bmatrix}$. Let $c_A := c_M|_A$. Trying to find a suitable solution x_O, we can proceed by induction and assume that the claim holds for all smaller instances. Thus, we can assume that we can also find a solutions x_A and x_B in polynomial time which are feasible for the inputs $|A|, B_M, c_A$ and $|B|, B_N, c_N$ and that we have $c_A(x_A) \leq 2c_A(x_A^\star)$ and $c_B(x_B) \leq 2c_B(x_B^\star)$, where $x_A^\star$ and $x_B^\star$ are optimal solutions for the respective instances. To continue making progress on this topic, it might prove helpful to first solve the problem for the case of two graphic matroids, as we do have a graph-interpretation of the input-matroids for this case.

Given these thoughts, we leave solving the question even for the case of the two-sum open for now.

Bibliography

[1] D. Adjiashvili. Beating Approximation Factor Two for Weighted Tree Augmentation with Bounded Costs. *ACM Transactions on Algorithms (TALG)*, 15(2):1–26, 2018.

[2] J. A. Bondy, U. S. R. Murty, et al. *Graph Theory with Applications*, volume 290. Macmillan London, 1976.

[3] T. Bosman, D. Frascaria, N. Olver, R. Sitters, and L. Stougie. Fixed-Order Scheduling on Parallel Machines. In *Integer Programming and Combinatorial Optimization: 20th International Conference, IPCO 2019, Ann Arbor, MI, USA, May 22-24, 2019, Proceedings 20*, pages 88–100. Springer, 2019.

[4] J. Byrka, F. Grandoni, and A. J. Ameli. Breaching the 2-Approximation Barrier for Connectivity Augmentation: A Reduction to Steiner Tree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 815–825, 2020.

[5] F. Cecchetto, V. Traub, and R. Zenklusen. Bridging the Gap Between Tree and Connectivity Augmentation: Unified and Stronger Approaches. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 370–383, 2021.

[6] C. Chekuri and S. Khanna. Approximation Algorithms for Minimizing Average Weighted Completion Time. In *Handbook of scheduling: Algorithms, models, and performance analysis*, pages 11–1. CRC press, 2004.

[7] C. Chekuri and R. Motwani. Precedence Constrained Scheduling to Minimize Sum of Weighted Completion Times on a Single Machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[8] J. Cheriyan and Z. Gao. Approximating (Unweighted) Tree Augmentation via Lift-and-Project, Part II. *Algorithmica*, 80:608–651, 2018.

[9] F. A. Chudak and D. S. Hochbaum. A Half-Integral Linear Programming Relaxation for Scheduling Precedence-Constrained Jobs on a Single Machine. *Operations Research Letters*, 25(5):199–204, 1999.

[10] N. Cohen and Z. Nutov. A $(1 + \ln 2)$-Approximation Algorithm for Minimum-Cost 2-Edge-Connectivity Augmentation of Trees with Constant Radius. *Theoretical Computer Science*, 489:67–74, 2013.

[11] E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov. On the Structure of the System of Minimum Edge Cuts in a Graph. *Issledovaniya po Diskretnoi Optimizatsii*, pages 290–306, 1976.

[12] I. Dinur, S. Khot, G. Kindler, D. Minzer, and M. Safra. Towards a Proof of the 2-to-1 Games Conjecture? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 376–389, 2018.

[13] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. Uma, and J. Wein. Techniques for Scheduling with Rejection. *Journal of Algorithms*, 49(1):175–191, 2003.

[14] S. Fiorini, M. Groß, J. Könemann, and L. Sanità. Approximating Weighted Tree Augmentation via Chvátal-Gomory Cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 817–831. SIAM, 2018.

[15] L. Fleischer, K. Jain, and D. P. Williamson. Iterative Rounding 2-Approximation Algorithms for Minimum-Cost Vertex Connectivity Problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.

[16] G. N. Frederickson and J. Jájá. Approximation Algorithms for Several Graph Augmentation Problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[17] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson. Improved Approximation Algorithms for Network Design Problems. Technical report, Cornell University Operations Research and Industrial Engineering, 1995.

[18] M. X. Goemans and D. P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[19] R. L. Graham. Bounds on Multiprocessor Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[20] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[21] F. Grandoni, C. Kalaitzis, and R. Zenklusen. Improved Approximation for Tree Augmentation: Saving by Rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 632–645, 2018.

[22] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. *Mathematics of operations research*, 22(3):513–544, 1997.

[23] K. Jain. A factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica*, 21:39–60, 2001.

[24] T. Jansen. Introduction to the theory of complexity and approximation algorithms. *Lectures on Proof Verification and Approximation Algorithms*, pages 5–28, 2006.

[25] R. M. Karp. *Reducibility Among Combinatorial Problems*. Springer, 2010.

[26] L. G. Khachiyan. A Polynomial Algorithm in Linear Programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[27] S. Khot. On the Power of Unique 2-Prover 1-Round Games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

[28] S. Khot, D. Minzer, and M. Safra. On Independent Sets, 2-to-2 Games, and Grassmann Graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 576–589, 2017.

[29] S. Khot and O. Regev. Vertex Cover Might Be Hard to Approximate to within $2-\varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[30] J. Könemann, N. Olver, K. Pashkovich, R. Ravi, C. Swamy, and J. Vygen. On the Integrality Gap of the Prize-Collecting Steiner Forest LP. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2017.

[31] G. Kortsarz, R. Krauthgamer, and J. R. Lee. Hardness of Approximation for Vertex-Connectivity Network Design Problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[32] G. Kortsarz and Z. Nutov. A simplified 1.5-Approximation Algorithm for Augmenting Edge-Connectivity of a Graph from 1 to 2. *ACM Transactions on Algorithms (TALG)*, 12(2):1–20, 2015.

[33] G. Kortsarz and Z. Nutov. LP-Relaxations for Tree Augmentation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.

[34] R. Krishnaswamy, S. Li, and S. Sandeep. Constant Approximation for k-Median and k-Means with Outliers via Iterative Rounding. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 646–659, 2018.

[35] S. Li. Scheduling to Minimize Total Weighted Completion Time via Time-Indexed Linear Programming Relaxations. *SIAM Journal on Computing*, 49(4):FOCS17–409, 2020.

[36] A. Linhares, N. Olver, C. Swamy, and R. Zenklusen. Approximate Multi-Matroid Intersection via Iterative Refinement. In *Integer Programming and Combinatorial Optimization: 20th International Conference, IPCO 2019, Ann Arbor, MI, USA, May 22-24, 2019, Proceedings 20*, pages 299–312. Springer, 2019.

[37] E. Minieka. Finding The Circuits of a Matroid. *Journal of Research of the National Bureau of Standards - B. Mathematical Sciences*, 80B, No.3, 1976.

[38] A. Munier, M. Queyranne, and A. S. Schulz. Approximation Bounds for a General Class of Precedence Constrained Parallel Machine Scheduling Problems. In *Integer Programming and Combinatorial Optimization: 6th International IPCO Conference Houston, Texas, June 22–24, 1998 Proceedings 6*, pages 367–382. Springer, 1998.

[39] H. Nagamochi and T. Ibaraki. An Approximation for Finding a Smallest 2-Edge-Connected Subgraph Containing a Specified Spanning Tree. *Lecture notes in computer science*, pages 31–40, 1999.

[40] J. G. Oxley. *Matroid Theory*, volume 3. Oxford University Press, USA, 2006.

[41] R. Raz and S. Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484, 1997.

[42] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[43] A. S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds. In *Integer Programming and Combinatorial Optimization: 5th International IPCO Conference Vancouver, British Columbia, Canada, June 3–5, 1996 Proceedings 5*, pages 301–315. Springer, 1996.

[44] A. S. Schulz and M. Skutella. Random-Based Scheduling New Approximations and LP Lower Bounds. In *Randomization and Approximation Techniques in Computer Science: International Workshop RANDOM'97 Bologna, Italy, July 11–12, 1997 Proceedings 1*, pages 119–133. Springer, 1997.

[45] A. S. Schulz and M. Skutella. The Power of α-Points in Preemptive Single Machine Scheduling. *Journal of Scheduling*, 5(2):121–133, 2002.

[46] P. D. Seymour. Decomposition of Regular Matroids. *Journal of combinatorial theory, Series B*, 28(3):305–359, 1980.

[47] D. Shabtay, N. Gaspar, and M. Kaspi. A Survey on Offline Scheduling with Rejection. *Journal of scheduling*, 16:3–28, 2013.

[48] J. B. Sidney. Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs. *Operations Research*, 23(2):283–298, 1975.

[49] R. Sitters and L. Yang. A $(2+\varepsilon)$-Approximation for Precedence Constrained Single Machine Scheduling with Release Dates and Total Weighted Completion Time Objective. *Operations Research Letters*, 46(4):438–442, 2018.

[50] W. E. Smith. Various Optimizers for Single-Stage Production. Technical report, California University Los Angeles Numerical Analysis Research, 1955.

[51] V. Traub and R. Zenklusen. A Better-Than-2 Approximation for Weighted Tree Augmentation. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–12. IEEE, 2022.

[52] V. Traub and R. Zenklusen. Local Search for Weighted Tree Augmentation and Steiner Tree. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3253–3272. SIAM, 2022.

[53] V. Traub and R. Zenklusen. A $(1.5+\varepsilon)$-Approximation Algorithm for Weighted Connectivity Augmentation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1820–1833, 2023.

[54] W. T. Tutte. A Homotopy Theorem for Matroids. I. *Transactions of the American Mathematical Society*, 88(1):144–160, 1958.

[55] W. T. Tutte. A Homotopy Theorem for Matroids. II. *Transactions of the American Mathematical Society*, 88(1):161–174, 1958.

[56] J. D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[57] V. V. Vazirani. *Approximation Algorithms*, volume 1. Springer, 2001.

[58] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge university press, 2011.

[59] A. C.-C. Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227. IEEE Computer Society, 1977.